

z/OS



UNIX System Services Planning

z/OS



UNIX System Services Planning

Note

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 381.

Sixth Edition, March 2004

This edition applies to Version 1 Release 5 of z/OS (5694-A01), to Version 1 Release 5 of z/OS.e (5655-G52), and to all subsequent releases and modifications until otherwise indicated in new editions.

This is a major revision of GA22-7800-04.

IBM welcomes your comments. A form for readers' comments may be provided at the back of this document, or you may address your comments to the following address:

International Business Machines Corporation
Department 55JA, Mail Station P384
2455 South Road
Poughkeepsie, NY 12601-5400
United States of America

FAX (United States & Canada): 1+845+432-9405

FAX (Other Countries):

Your International Access Code +1+845+432-9405

IBMLink™ (United States customers only): IBMUSM10(MHVRCFS)

Internet e-mail: mhvrcfs@us.ibm.com

World Wide Web: <http://www.ibm.com/servers/eserver/zseries/zos/webqs.html>

If you would like a reply, be sure to include your name, address, telephone number, or FAX number.

Make sure to include the following in your comment or note:

- Title and order number of this document
- Page number or topic related to your comment

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 1996, 2004. All rights reserved.

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

Contents

Figures	xv
Tables	xvii
About this document	xix
Who should use this document?	xix
How to use this document.	xix
Where to find more information.	xix
Summary of changes	xxiii
Chapter 1. Introduction to z/OS UNIX	1
z/OS UNIX support.	1
Interaction with elements and features of z/OS	2
Workload Manager (WLM)	2
System Management Facilities (SMF)	4
C/C++	4
Language Environment	4
DFSMS/MVS	4
Security Server (RACF)	4
Resource Measurement Facility (RMF)	5
System Display and Search Facility (SDSF).	5
Time Sharing Options Extensions (TSO/E)	5
z/OS Communications Services (TCP/IP Services)	5
ISPF	6
IBM Library Server	6
Network File System (NFS).	6
zSeries File System (zFS)	6
Hardware considerations.	6
Workstation connections	6
What you can do with z/OS UNIX System Services Application Services	8
Application programmers.	8
Types of applications	9
Administrative tasks using the ISPF shell.	9
Chapter 2. Installing z/OS UNIX	11
Overview	11
Methods of installation	11
ServerPac customers	11
CBPDO customers	11
Establishing an /etc file system for a new release	12
Chapter 3. Customizing z/OS UNIX	15
Overview	15
Using the z/OS TCP/IP configuration wizard on the web.	15
Setting up kernel services	15
Minimum mode.	15
Full function mode	16
Setting up for full function mode	16
Evaluating virtual storage needs	17
Extended Common Service Area (ECSA)	17
Extended System Queue Area (ESQA)	17
Prioritizing kernel work on your system	20

Running in goal mode	21
Defining BPXPRMxx parmlib members in IEASYSxx	23
Customizing the BPXPRMxx parmlib members	24
Defining file systems	27
Defining system limits	30
Defining system features	35
AUTHPGMLIST	39
Customizing other parmlib members	39
ALLOCxx parmlib member to control allocation requests	39
COFVLFxx parmlib member to activate RACF classes	39
CTnBPXxx parmlib member to control tracing	40
IEADMR00 parmlib member to gather dump data	42
SMFPRMxx parmlib member to specify timeouts	42
Customizing /etc	42
Customizing the OMVS cataloged procedure to run the kernel initialization program	42
Running a physical file system in a colony address space	43
How to start colonies.	43
How to start colonies outside of JES	43
Running a temporary file system in a colony address space	45
Steps for creating a cataloged procedure for a temporary file system	45
Enabling certain TSO/E commands to z/OS UNIX users.	46
Setting up the REXX parameter modules	48
Steps for adding the syscall command to the default environment	48
Checking for setup errors	49
 Chapter 4. Using msys for Setup for z/OS UNIX.	51
Overview	51
Who should use msys for Setup?	51
What is a z/OS UNIX customization task?	51
Preparing to use msys for Setup	52
Using msys for Setup for z/OS UNIX customization	52
 Chapter 5. Establishing UNIX security	55
Overview	55
In this chapter	55
Preparing for RACF	56
Preparing RACF	56
Managing group identifiers and user identifiers (GIDs and UIDs).	59
Setting up users and groups	60
Activating supplemental groups	60
Defining z/OS UNIX users to RACF	60
Setting up default OMVS segments	63
Supporting case-sensitive user IDs	65
Security implications of programs running in the HFS.	65
Authority checks	66
Obtaining security information about a group	67
Obtaining security information about users.	67
Setting up field-level access for the OMVS segment of a user profile	68
Defining group identifiers (GIDs)	69
Defining user identifiers (UIDs)	69
Defining protected user IDs	70
Defining the terminal group name	70
Managing user and group assignments	71
Upper limits for GIDs and UIDs	72
Creating z/OS UNIX groups	73

Assigning superuser attributes	74
What can superusers do?	74
Defining superusers with appropriate privileges	74
Using UNIXPRIV class profiles	75
Assigning superuser privileges	77
Allowing z/OS UNIX users to change file ownerships	78
Using the BPX.SUPERUSER resource in the FACILITY class.	79
Steps for setting up BPX.SUPERUSER	79
Changing a superuser from UID(0) to a unique nonzero UID	80
Switching in and out of superuser authority	82
Assigning a UID of 0.	83
Setting up the BPX.* FACILITY class profiles.	83
Security requirements for ServerPac and CBPDO installation	86
If you use uppercase group and user IDs	87
If you use mixed-case group and user IDs	88
If you have problems with names such as UUCP, UUCPG, and TTY	88
Defining cataloged procedures to RACF	89
Controlling access to files and directories	89
Setting classes for a user's process	90
Accessing files	91
Changing the permission bits for a file	92
Changing the owner or group for a file	92
Creating a set-user-ID or set-group-ID executable file.	92
Protecting data	93
Obtaining security information for a file	93
Using access control lists (ACLs)	95
ACLs and ACL entries	95
Managing ACLs	96
Using ACLs in a sysplex	100
Using security labels	100
Symbolic link restrictions	101
Using multilevel security	101
Security labels for zFS files and directories	101
Auditing access to files and directories.	101
Specifying file audit options	102
Using sanction lists	102
Formatting rules for sanction lists.	103
Steps for creating a sanction list	104
Steps for activating the sanction list.	105
Maintaining the security level of the system	106
Steps for maintaining the security level of the system	107
Defining the OMVSAPPL profile for the APPL class	107
Setting up TCP/IP security	108
Selecting a security level for the system	108
Chapter 6. Managing the hierarchical file system.	109
Overview	109
In this chapter.	109
Hierarchical file system concepts.	109
Required file system structure	110
Command differences due to symbolic links	111
Recommended file system structures for user directories and files.	111
Using the Network File System (NFS)	112
Creating a hierarchical file system	112
Using uppercase and lowercase letters in filenames and pathnames.	113
Allocating an HFS data set for the root file system	113

Defining the root file system.	114
What happens when file systems are mounted?	114
Mounting file systems	116
Restrictions on mounting file systems	117
Managing file systems.	118
Reducing the size of the file system.	118
Increasing the size of the file system	118
Removing unnecessary files from directories	119
Improving accesses to file systems	119
Unmounting file systems	119
Read-only BPAM access	119
Mounting your root HFS for execution	120
Deciding how to mount your root HFS for execution.	120
Leaving the root HFS mounted in read/write mode	120
Post-installation actions for mounting the root HFS in read-only mode	121
Mounting the root HFS in read-only mode	122
Customizing cron, uucp, and mail utilities for a read-only root HFS	122
Customizing the cron and uucp utilities	123
Customizing the mail utility	124
Remounting a mounted file system	126
Remounting with zFS aggregates	126
Copying the file system	126
Backing up HFS data sets	126
Ways to back up HFS data sets	127
Creating the user file systems	129
Making user file systems available	129
Using direct mount	131
Using file locks	134
Creating special files	135
Pseudoterminal files	136
Null file	136
File descriptor files	137
UNIX domain socket name special file	137
System console file	137
Handling file system failures	137
Restoring the root file system	137
Recovering from file system problems with the root	137
Installing service into the HFS	139
Example of installing service	141
Transporting the HFS from the driving system to the target system	141
Installing service into /etc	142
Installing products into the HFS	142
Chapter 7. Using the zSeries File System (zFS)	145
Overview	145
How does zFS differ from HFS?	145
When would you want to use zFS?	145
zFS and shared sysplex	145
Mounting considerations	146
Mount behaviors	146
Read-only BPAM access	146
Chapter 8. Using the automount facility	147
Overview	147
In this chapter.	147
Automounting NFS file systems	147

	Automounting zFS file systems	147
	Automounting in a sysplex	148
	How does the automount facility work?	148
	Using the automount facility.	148
	/etc/auto.master	149
	MapName	149
	Steps for setting up the automount facility	150
	What happens when you start the automount facility from the shell?	151
	Naming specific directories	153
	Changing which data sets get automounted	154
	Stopping the automount facility	154
	 Chapter 9. Shared HFS in a sysplex	155
	Overview	155
	In this chapter.	155
	What does shared HFS mean?	155
	How the end user views the HFS.	156
	Summary of new HFS data sets	156
	Comparing file systems in single system pre-OS/390 UNIX V2R9 and OS/390	
	UNIX V2R9 or later environments.	157
	File systems in single system pre-OS/390 UNIX V2R9 environments	158
	File systems in single system OS/390 UNIX V2R9 or later environments . .	159
	File systems in OS/390 UNIX V2R9 or later sysplex environments	160
	Procedures for establishing shared HFS in a sysplex	161
	Creating the sysplex root HFS data set	161
	Creating the system-specific HFS data sets	162
	Mounting the version HFS	162
	Using the automove system list (SYSLIST)	164
	Creating an OMVS couple data set (CDS)	164
	Customizing BPXPRMxx for shared HFS.	166
	Sysplex scenarios showing shared HFS capability	168
	Scenario 1: First system in the sysplex	168
	Scenario 2: Multiple systems in the sysplex using the same release level	171
	Scenario 3: Multiple systems in a sysplex using different release levels	174
	Keeping automount policies consistent on all systems in the sysplex.	176
	Keeping your automount policy consistent on all systems	177
	Moving file systems in a sysplex	177
	Shared HFS implications during system failures and recovery	178
	Managing the movement of data	179
	Shared HFS implications during a planned shutdown of z/OS UNIX	180
	State of file systems after shutdown.	181
	File system initialization	181
	Locking files in the sysplex	182
	Using distributed BRLM	183
	Mounting file systems using symbolic links	184
	Examples	184
	Mounting file systems using NFS client mounts	185
	File system availability.	185
	Minimum setup required for file system availability	185
	Situations that can interrupt availability.	187
	Tuning z/OS UNIX performance in a sysplex	187
	DFS considerations.	188
	 Chapter 10. Customizing the shells and utilities	189
	Overview	189
	Connecting to the shell	189

Invoking the shell automatically under TSO/E	189
Determining the CPU time limit	190
Supplying an alternative shell	190
Customizing the z/OS UNIX shells	191
Customizing the shell environment variables	191
Customizing the RACF user profile	192
Customizing files for the z/OS shell	192
Customizing files for the tcsh shell	204
Configuration files	205
Installing books and a bookshelf for the OHELP command	206
Enabling the man pages	206
Setting up for mesg, talk, write, and UUCP	207
Customizing c89, cc, and c++ (cxx) compilers	207
Using non-default high-level qualifiers	207
Using a system that does not have UNIT=SYSDA	208
Selecting C/C++ compilers	208
Targeting an OS/390 release earlier than the current one	209
Customizing the terminfo database	210
Steps for defining terminals or workstations for a terminfo database	210
Re-creating the terminfo database	211
Customizing electronic mail	211
For the z/OS shell	211
For the tcsh shell	212
 Chapter 11. Customizing for your national code page in the shell	213
Overview	213
In this chapter.	213
Setting up your national code page	213
Steps for setting up your national code page	213
Customizing for Japanese and Simplified Chinese	216
Steps for customizing the login file for the z/OS shell	216
Steps for customizing the login file for the tcsh shell.	216
Steps for displaying messages	217
Steps for activating MVS Message Service (MMS)	217
Concatenating target libraries to ISPF	218
PROFILE PLANGUAGE and the OMVS command	218
 Chapter 12. Configuring the UNIX-to-UNIX copy program (UUCP)	221
Overview	221
What is UUCP?	221
Transferring files	221
Executing commands from a remote location	221
Custom applications	221
UUCP commands and daemons	222
UUCP directories and files	222
The UUCP communications network	223
Securing your system	223
The public UUCP directory	224
Execute permissions	225
Configuring your local system	225
Determine your local system name	226
Add an entry to the permissions file.	226
Define the group ID and the user ID to RACF	226
Configuring communication with remote systems	228
Obtain information about remote systems.	228
Create or edit configuration files	229

Compile the configuration files	238
Create working directories for the local and remote systems	238
Schedule periodic UUCP transfers with cron	239
Testing the connection	241
Problem determination for connection problems	241
Contacting the remote site	242
Calling system login	242
Maintaining UUCP	242
Cleaning up UUCP files	242
Displaying information about recorded UUCP events	243
Notifying remote systems about password changes	243
 Chapter 13. Using Enhanced ASCII functionality	 245
Overview	245
In this chapter.	245
Limitations of Enhanced ASCII.	245
Setting up Enhanced ASCII.	246
Steps for setting up Enhanced ASCII	246
 Chapter 14. Managing operations	 249
Overview	249
In this chapter.	249
Ending processes	249
Steps for ending a specified process	250
Ending threads	251
Ending threads with the MODIFY command.	251
Planned shutdowns using F BPXOINIT,SHUTDOWN=...	252
Steps for shutting down z/OS UNIX using F BPXOINIT,SHUTDOWN=...	252
Partial shutdowns (for JES2 maintenance)	254
File system shutdown	255
Planned shutdowns using F OMVS,SHUTDOWN	256
What F OMVS,SHUTDOWN does	256
Steps for shutting down z/OS UNIX using F OMVS,SHUTDOWN	257
Dynamically changing the BPXPRMxx parameter values	259
Dynamically changing certain BPXPRMxx parameter values.	259
Dynamically switching to different BPXPRMxx members	260
Dynamically adding FILESYSTYPE statements in BPXPRMxx	261
Steps for activating the HFS file system for the first time	261
Activating a single sockets file system for the first time.	262
Activating a multiple sockets file system for the first time with Common INET (CINET)	263
Increasing the MAXSOCKETS value	263
Adding another sockets file system to an existing Common INET (CINET) configuration	264
Tracing events	265
Tracing events in z/OS UNIX	265
Tracing DFSMS/MVS events	265
Re-creating problems for IBM service	266
Displaying the status of the kernel or process	266
Displaying the status of BPXPRMxx parmlib limits	268
Taking a dump of the kernel and user processes	268
Displaying the kernel address space	269
Displaying process information	269
Displaying global resource information.	270
Preallocating a sufficiently large dump data set	270
Taking dumps	270

Reviewing dump completion information	271
Recovering from a failure	271
z/OS UNIX system failure	271
File system type failure	271
File system failure	272
Recovery of DCE components.	272
Managing Interprocess Communication (IPC)	272
Chapter 15. Managing processing for z/OS UNIX	275
Overview	275
Controlling printing	275
Designating printers	275
Setting up default printers	275
Controlling output print separators	276
Controlling code page conversion	276
Considerations for data conversion	277
Customizing code page conversion	278
Managing z/OS UNIX in relation to other processing	278
JES2 processing	278
JES3 processing	279
Applications processing	279
Accessing the Language Environment run-time library	279
Enabling the fastpath support for system authorization facility (SAF)	281
Determining problem causes	281
Abends	281
Return codes and reason codes	282
Messages	282
Component identifiers	282
Dump formatting	282
Diagnosing problems in an IBM-supplied z/OS UNIX program	283
Diagnosing problems in application programs	283
Diagnosing hangs during z/OS UNIX initialization.	284
Chapter 16. Managing a temporary file system (TFS)	285
Overview	285
In this chapter.	285
Maximum file size	285
Mounting the TFS	285
Steps for mounting the TFS.	286
Unmounting the TFS	287
Using a TFS in a shared HFS environment	288
Read-only BPAM access	288
Chapter 17. Setting up for daemons	289
Overview	289
In this chapter.	289
Comparing UNIX security and z/OS UNIX security	290
Establishing the correct level of security for daemons	291
UNIX level	292
z/OS UNIX level	292
Customizing the system for IBM-supplied daemons	295
Defining modules to program control	295
Setting up HFS control	297
Defining HFS files as APF-authorized programs	298
Defining HFS files as shared library programs	299
Handling dirty address spaces.	299

Using enhanced program security	300
Customizing the system for IP-supplied daemons.	302
Steps for customizing the system for IP-supplied daemons	302
Customizing the IBM-supplied daemons	302
Customizing the inetd daemon.	302
Customizing the uucpd daemon	303
Customizing the rlogind daemon	304
Customizing the cron daemon	305
Starting and restarting daemons	305
Starting a daemon from the shell.	305
Setting up security procedures for daemons.	308
Steps for setting up security procedures for daemons	308
Giving daemon authority to vendor-written programs	309
Tracking down problems when setting up daemons and servers	310
Verifying the user OMVS segment	310
Verifying the group OMVS segment	311
Verifying that the sticky bit is on	312
Using external links to access MVS load libraries.	313
Finding modules that were not defined to program control	313
Checking the daemon authority	315
Checking the server setup	315
Setting up for rlogin.	317
Steps for setting up for rlogin	317
Problem determination	318
 Chapter 18. Preparing security for servers	319
Overview	319
Designing security for servers	319
Setting up threads and security	319
Checking authority to use protected resources	321
Limitations of RACF client ACEE support.	322
Documenting the security requirements	322
Establishing the correct level of security for servers	322
UNIX level: BPX.SERVER is not defined	322
z/OS UNIX level: BPX.SERVER is defined	323
RACF with enhanced program security, BPX.SERVER, and BPX.MAINCHECK	323
BPX.SERVER.	323
Defining servers to use thread-level security	324
Server setup	324
Defining servers to process users without passwords	326
 Chapter 19. Monitoring the z/OS UNIX environment.	329
Overview	329
Reporting on activities using SMF records	329
Using SMF Record Type 30.	329
Preventing SMF Record Type 34 and 35 for SUBSYS OMVS	330
Using SMF Record Type 74.	330
Using SMF Record Type 80.	330
Using SMF Record Type 92.	330
Monitoring process activity	331
Using installation exits.	331
 Chapter 20. Tuning performance	333
Overview	333
Adjusting storage size	333

Using DASD cache	333
Improving performance of run-time routines	333
Placing SCEERUN in the link pack area	333
Placing SCEERUN in the link list	334
Managing the run-time library with RTLS	334
Managing the run-time library in STEPLIBs	334
Improving compiler performance	335
Putting compiler load modules into LPA	335
Caching RACF user and group information in VLF	335
Moving HFS executables into the link pack area	336
Using the shared library extended attribute	337
Tuning limits in parmlib	338
Monitoring BPXPRMxx parameter limits	338
Tuning process activity	339
Controlling use of ESQA	339
Enabling nice(), setpriority(), and chpriority() support	340
Making sure that the sticky bit for the z/OS shell is on	341
Improving the z/OS shell performance	341
Setting environment variables	342
Avoiding use of STEPLIBs	342
Exporting specific STEPLIBs	343
Improving performance of file systems	343
The /tmp directory	344
Caching frequently-read files	345
Improving performance of security checking	346
OMVS command and TSO/E response time.	346
Improving the performance of the make utility	346
 Chapter 21. Setting up for sockets	 347
Overview	347
Choosing between INET or CINET	348
Setting up for INET	349
Customizing BPXPRMxx for INET	349
Setting up for CINET	349
The internal routing table.	351
Transport drivers.	352
Limitations of IP configurations using CINET	352
Customizing BPXPRMxx for CINET	352
Using specific transports under CINET.	355
Format of resolver configuration data	357
Host information	357
Service information	357
Protocol information	358
Resolver information	358
 Chapter 22. Managing accounting work	 359
Overview	359
MVS accounting	359
Assigning account numbers for forked address spaces.	359
Modifying the accounting information for the OMVS and BPXOINIT address space	360
IEFUAV — user account validation exit	361
IEFUJI — job initiation exit	361
IEFUJV — job validation exit	363
IEFUSI — step initiation exit	363
Generating jobnames for OMVS address spaces	364

Chapter 23. z/OS UNIX System Services Parallel Environment	365
Overview	365
Setting up the partition daemon	365
Customizing X-Windows resources	365
Customizing your code page	365
Using Parallel Environment	365
Setting up the execution environment	366
Parallel Environment files	366
Appendix A. Commonly used environment variables	369
Appendix B. login and logout functions	373
FOMTLINP module for login function	373
FOMTLOUT module for logout function	376
Appendix C. Accessibility	379
Using assistive technologies	379
Keyboard navigation of the user interface.	379
z/OS Information.	379
Notices	381
Programming Interface Information	382
Trademarks.	383
Glossary	385
Index	441

Figures

1.	z/OS operating system with z/OS UNIX	2
2.	Creation of a new process by fork()	3
3.	OMVS UID and GID in RACF profiles	5
4.	Example of workstation and network connections	7
5.	BPXPRMXX parmlib member in SAMPLIB	25
6.	CTIBPX00 parmlib member	41
7.	Customized CTCBPX08 parmlib member	41
8.	Sample ISPF selection menu	47
9.	Logical view of the HFS for the end user	110
10.	Job to allocate the HFS data set.	114
11.	Mounting a file system	115
12.	JCL to allocate user HFS data sets	129
13.	Direct mount	130
14.	Automount facility	131
15.	JCL to allocate intermediate HFS data set	131
16.	Mounting the new intermediate HFS data set	132
17.	Creating a user's mount point directory	133
18.	Mounting the new HFS data set	134
19.	A pipe between two processes	135
20.	MKNOD TSO/E commands for a master-and-slave pair	136
21.	Preparation for installing service.	141
22.	Example of an /etc/auto.master file, /etc/u.map	149
23.	Example of a generic entry in a MapName file, /etc/u/map	149
24.	Follow-up steps when using the automount facility	152
25.	Specific entry in a MapName file	154
26.	Logical view of shared HFS for the end user	156
27.	BPXPRMxx for a single system before OS/390 UNIX V2R9 or later environments	158
28.	Single system before OS/390 UNIX V2R9	158
29.	BPXPRMxx parmlib member for single system: OS/390 UNIX V2R9	159
30.	Single system: OS/390 UNIX V2R9	160
31.	Sysplex root	161
32.	System HFS	162
33.	Version HFS	163
34.	COUPLExx parmlib member	166
35.	BPXPRMxx parmlib setup — HFS sharing	169
36.	HFS sharing in a sysplex	170
37.	Sharing HFS data sets: one version HFS and one BPXPRMxx for the entire sysplex	171
38.	Sharing HFS data sets: one version HFS and separate BPXPRMxx members for each system in the sysplex	172
39.	Sharing HFS data sets in a sysplex for OS/390 V2R9: multiple systems in a sysplex using the same release level.	173
40.	BPXPRMxx parmlib setup for multiple systems sharing HFS data sets and using different release levels	174
41.	Sharing HFS data sets between multiple systems using different release levels	175
42.	One BPXPRMxx parmlib member for multiple systems sharing HFS data sets and using different release levels	176
43.	Partial contents of IBM-supplied /samples/profile.	193
44.	Contents of /samples/.profile	197
45.	Contents of sample /etc/rc file	202
46.	Partial contents of IBM-supplied /samples/csh.login.	204
47.	Partial contents of IBM-supplied /samples/csh.cshrc	205
48.	A simple UUCP network.	223
49.	Console display for a CANCEL command	251

50.	GTFTRACE output for tracking modules for program control	314
51.	Output from command RLIST FACILITY BPX.DAEMON AUTHUSER	315
52.	Output from command RLIST FACILITY BPX.SERVER AUTHUSER	316
53.	Output from command RLIST SURROGAT BPX.SRV.ANONYMOS AUTHUSER	316
54.	Job for placing a program in the LPA	337
55.	Eliminating STEPLIB propagation	343
56.	Propagating only the Language Environment run-time library STEPLIB	343
57.	The /tmp directory on a mountable file system	345
58.	z/OS System using a single stack	347
59.	z/OS UNIX system using multiple stacks	348
60.	Multiple transport driver support	350
61.	Multiple transport driver support with two z/OS UNIX systems	351
62.	Partial extract of the services information	357

Tables

	1. Access to z/OS UNIX	8
	2. Task list for customization in full function mode.	16
	3. File system types.	27
	4. System-wide and process-level limits	30
I	5. Copying files from /samples to /etc	42
	6. Resource names in the UNIXPRIV class for z/OS UNIX privileges.	75
	7. File access types and permission bits	91
	8. ACL tasks and their associated commands	99
	9. Comparing read-only and read/write mode for the execution system's root HFS	120
	10. Required post-installation activities for mounting a read-only root HFS.	121
	11. HFS data sets that exist in a sysplex	157
I	12. Automove options supported by the MOUNT command	179
	13. Configuration files	206
	14. UUCP configuration files	229
	15. Escape characters usable in chat scripts	233
	16. Default printers	275
I	17. Options for the PARM parameter	287
	18. Comparing traditional UNIX, MVS, and z/OS UNIX security.	290
	19. Environment variables often used by the kernel	369

About this document

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

The z/OS Network File System and the z/OS Distributed File Service provide additional capability with z/OS UNIX.

Using this document, the people who run the installation will be able to do the following for z/OS UNIX:

- Plan a migration from an earlier release, if applicable
- Customize it
- Manage operations
- Manage processing by z/OS shell users and application programs
- Manage file systems
- Control security
- Monitor and tune performance
- Collect data for accounting

Who should use this document?

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

This document assumes the readers are familiar with z/OS systems and with the information for z/OS and its accompanying products.

For information about the features and concepts of z/OS UNIX, and for answers to many questions you may have, see our web site at:

<http://www.ibm.com/servers/eserver/zseries/zos/unix/>

How to use this document

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

Where to find more information

Where necessary, this document references information in other documents about the elements and features of z/OS. For complete titles and order numbers for all z/OS documents, see *z/OS Information Roadmap*.

Direct your request for copies of any IBM publication to your IBM representative or to the IBM branch office serving your locality.

There is also a toll-free customer support number (1-800-879-2755) available Monday through Friday from 6:30 a.m. through 5:00 p.m. Mountain Time. You can use this number to:

- Order or inquire about IBM publications
- Resolve any software manufacturing or delivery concerns
- Activate the program reorder form to provide faster and more convenient ordering of software updates

Softcopy publications

The z/OS UNIX library is available on the *z/OS Collection Kit*, SK2T-6700. This softcopy collection contains a set of z/OS and related unlicensed product documents. The CD-ROM collection includes the IBM Library Reader™, a program that enables customers to read the softcopy documents.

Softcopy z/OS publications are also available for web-browsing and PDF versions of the z/OS publications for viewing or printing using Adobe Acrobat Reader at:

<http://www.ibm.com/servers/eserver/zseries/zos/>

Accessing z/OS licensed documents on the Internet

z/OS licensed documentation is available on the Internet in PDF format at the IBM Resource Link™ Web site at:

<http://www.ibm.com/servers/resourceLink>

Licensed documents are available only to customers with a z/OS license. Access to these documents requires an IBM Resource Link user ID and password, and a key code. With your z/OS order you received a Memo to Licensees, (GI10-0671), that includes this key code.¹

To obtain your IBM Resource Link user ID and password, log on to:

<http://www.ibm.com/servers/resourceLink>

To register for access to the z/OS licensed documents:

1. Sign in to Resource Link using your Resource Link user ID and password.
2. Select **User Profiles** located on the left-hand navigation bar.

Note: You cannot access the z/OS licensed documents unless you have registered for access to them and received an e-mail confirmation informing you that your request has been processed.

Printed licensed documents are not available from IBM.

You can use the PDF format on either **z/OS Licensed Product Library CD-ROM** or IBM Resource Link to print licensed documents.

Using LookAt to look up message explanations

LookAt is an online facility that lets you look up explanations for most of the IBM messages you encounter, as well as for some system abends and codes. Using LookAt to find information is faster than a conventional search because in most cases LookAt goes directly to the message explanation.

You can use LookAt from the following locations to find IBM message explanations for z/OS elements and features, z/VM®, and VSE:

1. z/OS.e™ customers received a Memo to Licensees, (GI10-0684) that includes this key code.

- The Internet. You can access IBM message explanations directly from the LookAt Web site at <http://www.ibm.com/eserver/zseries/zos/bkserv/lookat/>.
- Your z/OS TSO/E host system. You can install code on your z/OS or z/OS.e systems to access IBM message explanations, using LookAt from a TSO/E command line (for example, TSO/E prompt, ISPF, or z/OS UNIX System Services running OMVS).
- Your Windows® workstation. You can install code to access IBM message explanations on the *z/OS Collection* (SK3T-4269), using LookAt from a Windows DOS command line.
- Your wireless handheld device. You can use the LookAt Mobile Edition with a handheld device that has wireless access and an Internet browser (for example, Internet Explorer for Pocket PCs, Blazer, or Eudora for Palm OS, or Opera for Linux handheld devices). Link to the LookAt Mobile Edition from the LookAt Web site.

You can obtain code to install LookAt on your host system or Windows workstation from a disk on your *z/OS Collection* (SK3T-4269), or from the LookAt Web site (click **Download**, and select the platform, release, collection, and location that suit your needs). More information is available in the LOOKAT.ME files available during the download process.

IBM Systems Center publications

IBM systems centers produce redbooks that can be helpful in setting up and using z/OS UNIX System Services. You can order these publications through normal channels, or you can view them with a web browser:

<http://www.redbooks.ibm.com>

These documents have not been subjected to any formal review nor have they been checked for technical accuracy, but they represent current product understanding (at the time of their publication) and provide valuable information on a wide range of z/OS UNIX topics. You must order them separately. A selected list of these documents is on the z/OS UNIX web site at:

<http://www.ibm.com/servers/eserver/zseries/zos/unix/bpxa1pub.html/>

z/OS UNIX porting information

There is a *Porting Guide* on the z/OS UNIX porting page: :

<http://www.ibm.com/servers/eserver/zseries/zos/unix/bpxa1por.html/>

You can read the *Porting Guide* from the web or download it as a PDF file that you can view or print using Adobe Acrobat Reader. The *Porting Guide* covers a range of useful topics, including: sizing a port, setting up a porting environment, ASCII-EBCDIC issues, performance, and much more.

The porting page also features a variety of porting tips, and lists porting resources that will help you in your port.

z/OS UNIX courses

For a current list of courses that you can take, go to:

<http://www.ibm.com/services/learning/>

You can also see your IBM representative or call 1-800-IBM-TEACH (1-800-426-8322).

z/OS UNIX home page

The z/OS UNIX home page on the World Wide Web has the latest technical news, customer stories, tools, and FAQs (frequently asked questions). You can visit it at:

<http://www.ibm.com/servers/eserver/zseries/zos/unix/>

Some of the tools available from the web site are ported tools, and some are home-grown tools designed for z/OS UNIX. All this code works in our environment at the time we make it available, but is not officially supported. Each tool has a README file that describes the tool and any restrictions on its use.

The simplest way to reach these tools is through the z/OS UNIX home page. From the home page, click on **Tools and Toys**.

The code is also available from **ftp://ftp.software.ibm.com/s390/zos/unix/** through **anonymous ftp**.

Restrictions

Because the tools are not officially supported, there are no known enhancements and no APARs can be accepted.

z/OS UNIX customization wizard

If you'd like help with customizing z/OS UNIX, then check out our Web-based wizard. You can access it at:

<http://www.ibm.com/servers/eserver/zseries/zos/wizards/>

This wizard builds two BPXPRMxx parmlib members; one with system processing parameters and one with file system statements. It also builds a batch job that does the initial RACF security setup for z/OS UNIX. Whether you are installing z/OS UNIX for the first time or are a current user who wishes to verify settings, you can use this wizard.

The wizard also allows sysplex users to build a single BPXPRMxx parmlib member to define all the file systems used by systems participating in shared HFS.

Discussion list

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

To subscribe to the mvs-oe discussion so you can receive postings, send a note to: **listserv@vm.marist.edu**

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

`subscribe mvs-oe first_name last_name`

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

Summary of changes

Summary of changes for GA22-7800-05 z/OS Version 1 Release 5

The document contains information previously presented in *z/OS UNIX System Services Planning*, GA22-7800-04 which supports z/OS Version 1 Release 4.

New information

A new section, “Planned shutdowns using F BPXOINIT,SHUTDOWN=...” on page 252 has been added. It accompanies “Planned shutdowns using F OMVS,SHUTDOWN” on page 256.

Changed information

“Steps for increasing the MAXSOCKETS value” on page 263 has been updated because it is no longer necessary to shut down TCP/IP.

Deleted information

The section “Using the built-in c89/cc/c++ utility for the z/OS shell” has been deleted because **c89/cc/c++** built-in commands are no longer available.

Moved information

The automount information that was previously in Chapter 6, “Managing the hierarchical file system,” on page 109 has been moved to a new chapter, Chapter 8, “Using the automount facility,” on page 147.

Migration information for z/OS UNIX is now in *z/OS Migration*.

z/OS Summary of Message and Interface Changes lists interface changes for z/OS UNIX.

This book has been enabled for the following z/OS library center advanced searches: tasks.

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

Starting with z/OS V1R2, you may notice changes in the style and structure of some content in this document—for example, headings that use uppercase for the first letter of initial words only, and procedures that have a different look and format. The changes are ongoing improvements to the consistency and retrievability of information in our documents.

Summary of changes for GA22-7800-04 z/OS Version 1 Release 4

The document contains information previously presented in *z/OS UNIX System Services Planning*, GA22-7800-04, which supports z/OS Version 1 Release 4.

New information

“Planned shutdowns using F BPXOINIT,SHUTDOWN=...” on page 252 has been added.

Summary of changes for GA22-7800-03 z/OS Version 1 Release 4

The document contains information previously presented in *z/OS UNIX System Services Planning*, GA22-7800-03, which supports z/OS Version 1 Release 3.

Moved information

“Setting up the BPX.* FACILITY class profiles” on page 83 has been moved to Chapter 5.

Deleted information

The section titled “How z/OS UNIX uses TCP/IP data set names” has been removed. Refer to *z/OS Communications Server: IP Configuration Reference* for more information about TCP/IP data sets.

Summary of changes for GA22-7800-02 z/OS Version 1 Release 3

The document contains information previously presented in *z/OS UNIX System Services Planning*, GA22-7800-01, which supports z/OS Version 1 Release 2.

New information

Chapter 4, “Using msys for Setup for z/OS UNIX,” on page 51 has been added.

Chapter 7, “Using the zSeries File System (zFS),” on page 145 has been added.

An appendix with z/OS product accessibility information has been added.

Summary of changes for GA22-7800-01 z/OS Version 1 Release 2

The document contains information previously presented in *z/OS UNIX System Services Planning*, GA22-7800-00, which supports z/OS Version 1 Release 1.

New information

Chapter 13, “Using Enhanced ASCII functionality,” on page 245 has been added.

Changed information

This book contains terminology, maintenance, and editorial changes, including changes to improve consistency and retrievability.

Chapter 1. Introduction to z/OS UNIX

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

- An application program interface (API). The application interface is composed of C interfaces. Some of the C interfaces are managed within the C Run-Time Library (RTL), and others access kernel interfaces to perform authorized system functions on behalf of the unauthorized caller.
- 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

z/OS UNIX has two shells, the z/OS shell and the tcsh shell. They are collectively called the z/OS UNIX shells.

z/OS UNIX support

z/OS UNIX, which responds to requests from programs and the z/OS UNIX shells, is made up of system and application services.

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

It also contains the code that was provided in the optional Shell and Utilities and the Debugger features prior to z/OS.

Interaction with elements and features of z/OS

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)

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

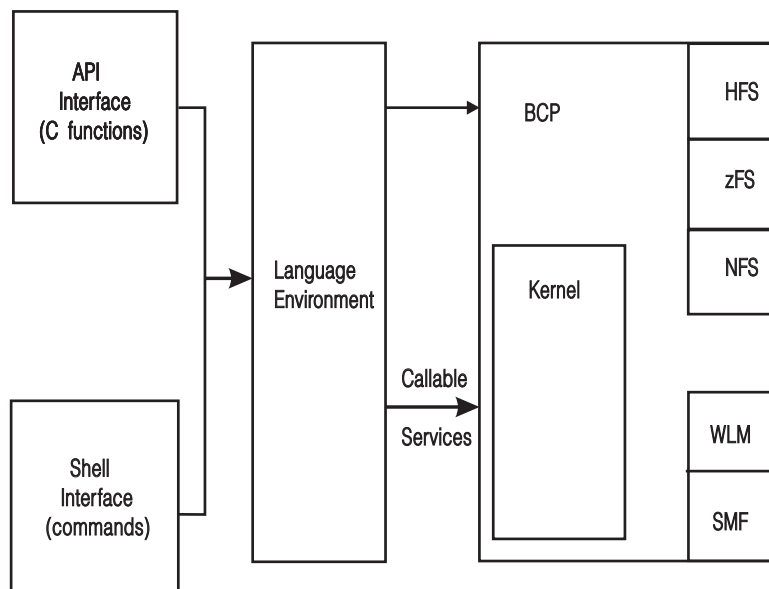


Figure 1. z/OS operating system with z/OS UNIX

Workload Manager (WLM)

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 while running in either goal mode or compatibility mode.

When programs issue `fork()` or `spawn()`, the BPXAS PROC found in SYS1.PROCLIB is used to provide a new address space. For a `fork()`, the system

copies one process, called the *parent process*, into a new process, called the *child process*. The *forked address space* is provided by WLM. See Figure 2.

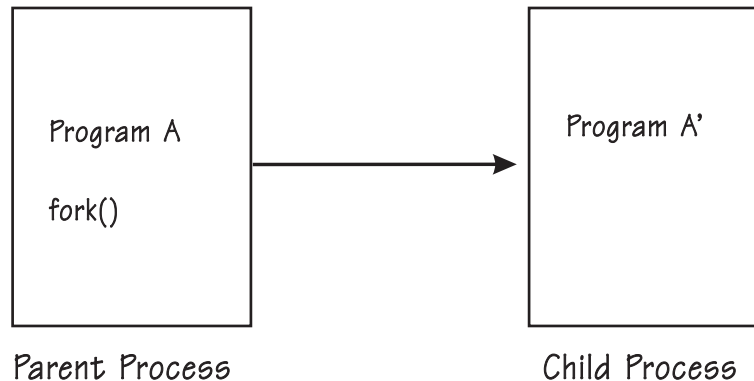


Figure 2. Creation of a new process by `fork()`

Processes can be created by a fork or spawn. Existing MVS address space types such as TSO, STC, Batch, and APPC can request z/OS UNIX services. When one of those address spaces makes its first request to the z/OS kernel, the kernel *dubs* the task; that is, it identifies the task as a z/OS UNIX process.

The types of processes are:

- User processes, which are associated with a user
- Daemon processes, which perform continuous or periodic systemwide functions, such as a Web server. Daemons are programs that are typically started when the operating system is initialized and remain active to perform standard services. Some programs are considered daemons that initialize processes for users even though these daemons are not long-running processes. Examples of daemons are:
 - **cron**, which starts applications at specific times
 - **inetd**, which provides service management for a network
 - **rlogind**, which starts a user shell session when one is requested, using a remote rlogin command

In similar systems, initialization usually starts a telnet daemon to perform terminal services. In addition to using a cron daemon, z/OS installations can use Operations Planning and Control/ESA (OPC/ESA) to set up a timed event.

Daemons are not restarted if they stop. You can restart them in any of several ways:

- The z/OS operator can restart daemons using a cataloged procedure. For more information, see “Starting and restarting daemons” on page 305.
- A system programmer can restart the daemon from a shell.
- You can use automation products such as NetView® to notice daemons terminating and then restart them using cataloged procedures.

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

For more information about threads, refer to the `pthread_create` service (BPXB1PTC) in *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.

System Management Facilities (SMF)

System management facilities (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.

Use the JWT value in the SMF parmlib SMFPRMxx to specify when to time-out an idle address space. SMF/WLM does the tracking. See “SMFPRMxx parmlib member to specify timeouts” on page 42 for more information.

C/C++

To compile C code using the **c89** command, or to compile C/C++ code using **cxx**, you need the C/C++ compiler that is available with z/OS.

Language Environment

To run a shell command or utility, or any user-provided application program written in C or C++, you need the C/C++ run-time library provided with Language Environment.

DFSMS/MVS

Data Facility System-Managed Storage/MVS (DFSMS/MVS) manages the data sets used for processing the hierarchical file system (HFS). These HFS data sets make up a file hierarchy, described in “Hierarchical file system concepts” on page 109.

A file hierarchy can consist 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.

To the MVS system, the file hierarchy is a collection of hierarchical file system (HFS) data sets. Each HFS data set is a mountable file system.

Security Server (RACF)

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

A user is identified by a UID, which is kept in the RACF user profile, and a GID, which is kept in the RACF group profile. See Figure 3.

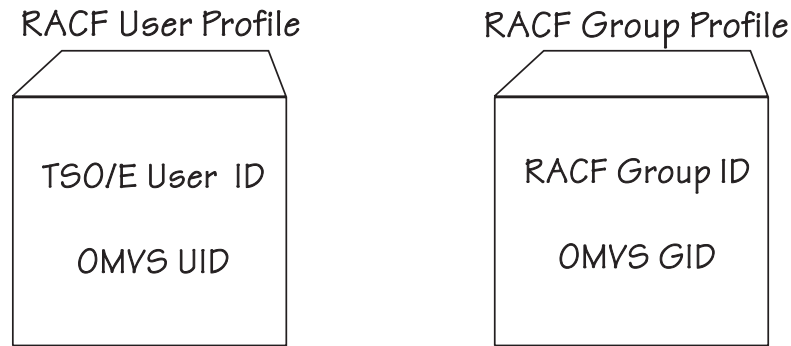


Figure 3. OMVS UID and GID in RACF profiles

For information on setting up default OMVS segments for users and groups, see “Setting up default OMVS segments” on page 63.

Resource Measurement Facility (RMF)

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

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

System Display and Search Facility (SDSF)

Shell users can enter TSO/E sessions and use SDSF to:

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

Time Sharing Options Extensions (TSO/E)

One way to enter the shell environment is 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/OS UNIX System Services User's Guide*.

z/OS Communications Services (TCP/IP Services)

Another way to enter the shell environment is using rlogin or telnet from a workstation in the TCP/IP network.

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

For information on multiple transport drivers, see Chapter 21.

ISPF

Users of ISPF can use the ISPF shell environment to create, edit, browse, and perform other functions for files and directories in the HFS.

IBM Library Server

You can invoke the online help facility with the TSO/E OHELP command and view online publications in BookManager format.

Network File System (NFS)

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

zSeries File System (zFS)

zSeries File System (zFS) is a UNIX file system that can be used, along with HFS. For more information, see Chapter 7.

Hardware considerations

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

Additional hardware considerations are:

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

Workstation connections

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

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

Figure 4 shows an example of workstation and network connections for the z/OS system with kernel services.

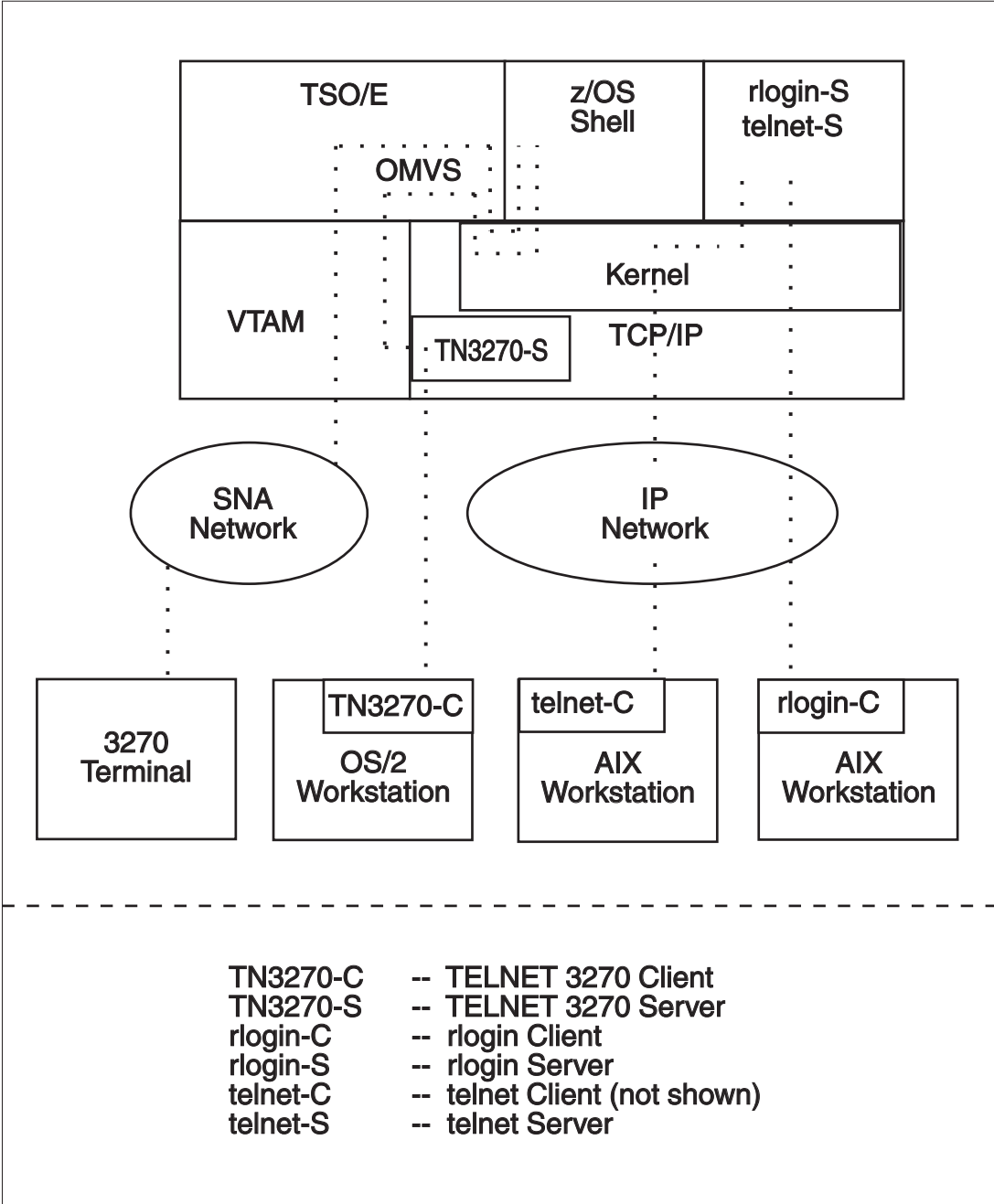


Figure 4. Example of workstation and network connections

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

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

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

- Line mode** Input is processed after you press <Enter>. Line mode is also called canonical mode.
- Raw mode** Each character is processed as it is typed. Raw mode is also called non-canonical mode.
- Graphical** A graphical user interface for X-Windows applications.

Table 1. Access to z/OS UNIX

Terminal	Software at the terminal	Connection to the host	Shell access	Supported modes
3270		Front-end processor such as 3174 or 3172	OMVS (TSO command)	Line
Workstation	3270 emulator (such as pc3270 or tn3270)	Front-end processor such as 3174 or 3172	OMVS (TSO command)	Line
	rlogin or telnet client	Front-end processor such as 3174 or 3172	rlogin or telnet	Line or raw
	X-Window server	Front-end processor such as 3174 or 3172	X-Window client	Graphical
X-terminal	rlogin or telnet client	Front-end processor such as 3174 or 3172	rlogin or telnet	Line or raw
	X-Window server	Front-end processor such as 3174 or 3172	X-Window client	Graphical

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

What you can do with z/OS UNIX System Services Application Services

With z/OS 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 and use terminology different from z/OS terminology. For example, they call virtual storage *memory*. The work done by their *system administrators* is handled by system programmers in z/OS systems. To help you understand these users, this document and its glossary indicate equivalent terms and phrases.

Application programmers

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

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

6. Use the application programs.

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

Types of applications

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

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

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

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

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

Administrative tasks using the ISPF shell

The ISPF shell is a panel interface that you can use instead of TSO/E commands or shell commands to perform certain tasks. The appropriate sections of this document mention for which tasks you can use this interface.

You can use the ISPF shell to work with the file system to do the following tasks:

- Display all mounted file systems
- Display the attributes of a mounted file system (such as total blocks, blocks in use, ddname)
- Make a file system (allocate an HFS data set)

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

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

See *z/OS UNIX System Services User's Guide* for more information about using the ISPF shell.

Chapter 2. Installing z/OS UNIX

Overview

This chapter describes the actions you take before you start to perform the installation steps outlined in *z/OS ServerPac: Installing Your Order* (for ServerPac users) and *z/OS Program Directory* (for CBPDO users).

I For migration information, go to *z/OS Migration*.

Methods of installation

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

It is important that you be familiar with the documents that come with those two installation methods. You will go to those documents to do the actual installing of z/OS UNIX, along with the other elements and features. The custom-built *z/OS ServerPac: Installing Your Order* describes the installation jobs that you run to replace an existing system or install a new one. For CBPDO users, the Program Directory describes how to use the SMP/E RECEIVE, APPLY, and ACCEPT commands to install your order. Both documents describe the installation verification procedures (IVPs) that you perform to ensure that your installation is proceeding successfully. They also contain some customization information.

ServerPac customers

For z/OS ServerPac customers, IBM delivers a single-root HFS. This HFS is unloaded when you do “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 z/OS UNIX.

Performing ServerPac installation requires that you be a superuser with UID(0) or have access to the BPX.SUPERUSER resource in the FACILITY class. See “Security requirements for ServerPac and CBPDO installation” on page 86 for a complete description of the security requirements necessary to perform your ServerPac install.

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

CBPDO customers

For customers who use the CBPDO (Custom-Built Product Delivery Option) 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.

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

You must be a superuser with UID(0) or have access to the BPX.SUPERUSER resource in the FACILITY class. See “Security requirements for ServerPac and CBPDO installation” on page 86 for a complete description of the security requirements necessary to perform your install.

Elements and features that install into the HFS are installed in both WAVE 1 and WAVE 2 of the CBPDO process and are listed in the z/OS Program Directory.

Setting up BPXOINIT as a started procedure

BPXOINIT is the started procedure that runs the initialization process. If you are using CBPDO, you have to set up BPXOINIT as a started procedure by adding it to either the RACF STARTED class or the RACF started procedures table, module ICHRINO3, as explained in “Preparing RACF” on page 56.

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

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

Establishing an /etc file system for a new release

The **/etc** file system 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 does not ship maintenance into **/etc** via SMP/E.

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

- If you do not have one, create it using instructions in “Customizing the z/OS UNIX shells” on page 191. For **/etc** file system files for other z/OS elements and features that install into HFS, see *z/OS Migration*. You might also have **/etc** file system impacts for non-z/OS products that you are installing. For that information, see migration and customization information for those products.
- If you already have an **/etc** file system, the **/etc** directory for the new z/OS system is based on a copy of the **/etc** file system for your existing system. You make this copy and migrate your existing **/etc** file system, following instructions in *ServerPac: Installing Your Order* (for those choosing ServerPac) and the Program Directory (for those choosing the CBPDO method of installation).

Because the configuration and customization data in your existing **/etc** file system might not be correct for the new system, you might need to make changes to the copy. For that information, turn to the information listed above.

Chapter 3. Customizing z/OS UNIX

Overview

If you plan to use msys for Setup to customize z/OS UNIX, go to Chapter 4.

Before customizing z/OS UNIX, you will need to decide whether you want to set up kernel services in minimum mode or full function mode.

Using the z/OS TCP/IP configuration wizard on the web

Recommendation: Use “z/OS UNIX Configuration Wizard”, a web-based tool, to help you set up z/OS UNIX in full function mode. This wizard begins with a series of interviews in which you will answer questions about your application environment and intentions regarding use of z/OS and TCP/IP. After you finish answering all of the interview questions, you ask the wizard to build the output. Then the wizard produces a checklist of steps for you to follow, as well as customized jobs and other data sets for you to use. Specifically, it builds two BPXPRMxx members and two HFS files and some RACF ALTUSER commands. The checklist of follow-on actions includes links to sections of this document and the IP Configuration Guide, thus eliminating the need to reference multiple documents.

Use this wizard to configure z/OS UNIX for the first time or to check and verify some of your configuration settings.

To use the wizard, go to:

<http://www.ibm.com/servers/eserver/zseries/zos/bkserv/wizards.html>

Setting up kernel services

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.

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

Minimum mode

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

If you specify OMVS=DEFAULT in the IEASYSxx parmlib member and then re-IPL, the kernel services start up in minimum mode and use the default values for all BPXPRMxx parmlib statements. See the BPXPRMxx section in *z/OS MVS Initialization and Tuning Reference* for information about the default values.

In minimum mode, a temporary file system named SYSROOT is used as the root file system. It is initialized and primed with a minimum set of files and directories. Any data written to this file system is not written to DASD. (See Chapter 16 for a

description of a temporary file system.) The temporary file system does not have any executables; that is, the shell will not be available. Do not install z/OS UNIX System Services Application Services in the TFS, because data will not be written to DASD.

If you do not set up the OMVS segments for the users and groups, you must set up the default UID/GIDs to use any of the kernel services. See “Setting up default OMVS segments” on page 63 for more information.

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

Full function mode

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

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

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

Setting up for full function mode

The following list identifies the tasks that you do to set up the kernel in full function mode and the sections where the tasks are described. It is important to follow the order of the tasks listed here.

Some tasks require superuser authority. In that case, that authority might be gained through authority to profiles defined in the UNIXPRIV class that grant certain superuser privileges to users who do not have superuser authority. For more information, see “Using UNIXPRIV class profiles” on page 75.

Table 2. Task list for customization in full function mode

Task	Topic
Evaluating virtual storage needs	17
Chapter 5, “Establishing UNIX security”	55
Prioritizing kernel work on your system	20
Defining BPXPRMxx parmlib members in IEASYSxx	23
Customizing the BPXPRMxx parmlib members	24
Customizing other parmlib members	39
Customizing the OMVS cataloged procedure to run the kernel initialization program	42
Running a physical file system in a colony address space	43
Enabling certain TSO/E commands to z/OS UNIX users	46
Setting up the REXX parameter modules	48
Setting up for the z/OS UNIX and tcsh shell environments	189, 191
Chapter 17, “Setting up for daemons”	289
Enabling the man pages	206

Table 2. Task list for customization in full function mode (continued)

Task	Topic
Setting up for rlogin	317
Chapter 18, "Preparing security for servers"	319
Creating the user file systems	129
Chapter 20, "Tuning performance"	333
Setting up TCP/IP security	108
Chapter 21, "Setting up for sockets"	347
Checking for setup errors	49

Evaluating virtual storage needs

The kernel services use storage based on expected use as defined by the BPXPRMxx parmlib member as well as by actual use. The following guidelines should help you avoid running out of storage.

Extended Common Service Area (ECSA)

Use of ECSA is based on the following formulas:

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

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

In addition to this ECSA usage,

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

Extended System Queue Area (ESQA)

Kernel services use ESQA in support of several functions. You can use formulas to predict some of the ESQA usage, but others can only be estimated. The maximum amount of ESQA consumed by z/OS UNIX shared memory functions as discussed in this section can be limited by the value specified in MAXSHAREPAGES.

The following functions consume ESQA:

1. Signaling uses SRBs to notify the target of a signal. Signaling frequency is usually not very high and the SRBs are short-lived. For most installations, additional ESQA does not need to be allocated in order to support signalling. If you run applications that use signals frequently, increase your ESQA allocation.
2. Using asynchronous socket services causes SRBs to be allocated. Allocate an additional 100KB of ESQA if there is heavy use of asynchronous socket services.

3. The following functions use an MVS service called IARVSERVER:

- ptrace(), debugger support
- shmat(), shared memory attach
- mmap(), memory map files
- fork() when fork is using Copy on Write (COW) mode
- dllload(), when it is loading a user-shared library program

For each real page of shared storage affected by IARVSERVER, RSM allocates a 32-byte anchor block in ESQA. For each virtual page connected to a shared real page, RSM allocates a 32-byte control block in ESQA. The following formulas and examples should help you to predict and limit ESQA usage:

- a. Using the dllload() function to load a user-shared library program (filename with a suffix of **.so**) causes IARVSERVER to set up a map between the user address space and the kernel shared library region data space. Only user-shared library programs will consume ESQA. Two parameters control the maximum number of ESQA that is used to load user-shared library storage. .

- MAXSHAREPAGES limits the total number of pages that can be consumed by shared memory applications. No more than 60% of the value that was specified can be used for the loading of user-shared library modules. To determine the maximum amount of ESQA that will be consumed while loading user-shared libraries, use the following formula:

$\text{MAXSHAREPAGES} \times .60 \times 32 = \text{Maximum number of bytes of ESQA}$

- SHRLIBMAXPAGES limits the maximum number of pages that the system can use for the loading of user-shared library modules. This parameter is intended to be used to limit user-shared library usage to a very small amount on systems where ESQA is very limited. For example, if the limit is set to a value of 100, then only 100 pages of user-shared library modules could be shared on the system concurrently. If this limit were to be set to 1, then user-shared library modules would not be shared on the system.

You can use this limit in conjunction with MAXSHAREPAGES to limit the amount of ESQA consumed by user-shared library modules. To estimate the usage of ESQA attributable to loading user shared libraries, use the following formula:

$\text{Bytes (in decimal) per user shared library object} \times (\text{expected number of connections} + 2) = \text{Total number of bytes needed for shared library objects}$

Rule: RSM must maintain a minimum of two blocks of storage in ESQA for each shared area that it manages via IARVSERVER, in addition to the blocks that are required for each connection. This is why 2 is always added to the number of connections. These two blocks of storage represent an anchor block and the data space storage area where the shared library module is loaded.

Divide this value by 4096 to get the total number of pages to be used, then multiply the result by 32 to find the estimated ESQA usage in bytes (again, in decimal).

Example: If a dllload() of a user-shared library program that is 500000 bytes in length is done from 50 address spaces:

$500000 \times (50 + 2) / 4096 \times 32 = 203125$

You will find that 203125 bytes of ESA, or approximately 200K, will be consumed.

- b. Shared memory, `shmat()`, is typically used by server address spaces to communicate with clients.

The `__IPC_MEGA` option enables applications to use large quantities of shared memory without the system overhead described later in this section. If you have applications taking advantage of this `__IPC_MEGA` support, you do not need to be concerned with the following calculations. As with user-shared libraries, the current usage can be calculated by multiplying the number of pages by the number of (connections plus 3) by 32.

Example: If a server that is not using 500000 bytes of shared memory and has 49 clients connected to it, the consumption of ESQA can be calculated as:

$$500000 * (49 + 3) / 4096 * 32 = 203125$$

You will find that 203125 bytes of ESQA, or approximately 200K, is needed.

The 49 plus 3 comes from 49 clients, 1 server, 1 anchor block, and 1 connection to a kernel data space that is used to manage the storage. Some servers like Lotus® Notes® use large amounts of shared memory that is shared by hundreds or thousands of clients. This can require large amounts of ESQA (up to one gigabyte).

- c. `mmap()` is typically used by a single process to map a file into virtual memory using the same sort of logic used by DIV (Data In Virtual). Used in this manner, each page of the file requires 3 RSM control blocks (anchor block, user page, and kernel data space page). Each additional user sharing an `mmap` page of a file will consume an additional control block.

The `__MAP_MEGA` option of `mmap()` enables applications to map very large files without the system overhead in ESQA. If you have applications using the `__MAP_MEGA` option, you do not need to be concerned with the above calculations. If you are not using `__MAP_MEGA` and issue `mmap()`, you can estimate the ESQA usage just as you would above for shared memory.

- d. `fork()` uses IARV SERV to capture the parent's pages for the child's use. Each page captured represents a requirement for three 32-byte RSM control blocks (parent, child, and an anchor block per page). Since the child usually issues the `exec` call soon after the `fork`, the ESQA used is short-term. This is countered by the probability that there are multiple forks going on concurrently. Again, the amount of required ESQA can be calculated by multiplying the size of the data area (in pages) to be copied by the number of concurrent forks + 3 by 32.

Example: Assuming the Language Environment Run-Time Library is not in LPA, a typical shell will have 5 MB of private to copy on fork. If there are, on average, 10 forks running concurrently, then the following ESQA is needed:

$$5\text{MB} * 256 \text{ pages/MB} * (10 \text{ forks} + 3) * 32 \text{ bytes/page}$$

You will find that 532 KB of ESQA is needed.

If the Run-Time Library does reside in LPA and each process has an average of 1 MB of private to copy, then:

$$1\text{MB} * 256 \text{ pages/MB} * (10 \text{ forks} + 3) * 32 \text{ bytes/page}$$

You will find that 106 KB of ESQA is needed.

- e. `ptrace()` uses captured storage to allow the debugger to map the program being debugged into private storage that the debugger can refer to

frequently. The amount of ESQA that is required can be calculated as the number of pages of storage required times the number of concurrent debug sessions +2 by 32.

Example: To calculate the amount of storage that will be needed if a programmer is debugging a 1MB program and a 200KB automatic data stack that are both captured:

$1.2\text{MB} * 256 \text{ pages/MB} * (1 \text{ debug session} + 2) * 32 \text{ bytes/page}$

The answer is 29KB of ESQA.

To predict the amount of ESQA required to support applications, you need to understand which applications use `shmat()`, `mmap()`, and `dllload()`. You need to approximate the amount of `fork()` and `dbx` debugger activity as well. Then plug your numbers into the equations for each function to determine the amount of ESQA needed.

In `BPXPRMxx`, specify the maximum number of shared storage pages that can be used on the `MAXSHAREPAGES` statement. By limiting the amount of shared storage pages used, `MAXSHAREPAGES` lets an installation control the amount of ESQA storage that is consumed by users.

This limit applies to the `mmap()`, `shmat()`, `ptrace()`, and `fork()` callable services, as well as the `dllload()` of user shared library modules (HFS modules with the `.so` suffix.)

The `fork()` and `ptrace()` callable services use shared storage pages to improve performance, as does a `dllload()` of shared library services. Because use of shared storage pages is not critical to completion of these functions, when the amount of shared storage pages in use reaches about 60% of the specified limit, these functions no longer use shared storage pages. The `mmap()` service continues to use the shared storage pages until the total resource consumption reaches about 80% of the limit. The `shmat()` callable service continues to use shared storage pages until the total resource consumption reaches the specified limit.

The `mmap()` and `shmat()` callable services return an out-of-memory condition when they can no longer obtain shared storage without exceeding their respective shared storage limits.

There is also a `FORKCOPY` parameter in `BPXPRMxx` that prevents `fork` from using the `IARVSERV` function.

Reducing the amount of ESQA needed to support servers

To reduce the excessive amounts of ESQA that are required to support a server like System Authorization Facility (SAF) that needs to access more than 2 GB of storage, you can use the following services:

- **`_map_init`**, which invokes the `map` service function.
- **`_map_service`**, which enables applications to create new data blocks and to specify which `map` area block is to be used to view the new data block.

Prioritizing kernel work on your system

This section discusses goal mode, which is a workload manager (WLM) mode for prioritizing kernel work in your system.

The `nice()` and `setpriority()` functions use definitions in `BPXPRMxx` for goals. These definitions are optional, but if they are not specified, the `nice()` and `setpriority()` functions do not change the performance level. The following are some reasons for enabling `nice()` and `setpriority()` functions:

- If you are running applications that require the ability to control the priority of different processes, you must define appropriate priority levels for the application to use. This is typically done for real-time systems that are dedicated to running a single application.
- 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.

For more information, see “Enabling `nice()`, `setpriority()`, and `chpriority()` support” on page 340.

Running in goal mode

Installations that run in goal mode can take the following steps to customize service policies in their WLM definition:

- Define a workload for kernel work.
- Define service classes for kernel work:
 - Define a service class for forked children. You should specify a number of performance periods. Performance periods for short-running work can be given response-time goals or percentage response-time goals. Performance periods for long-running work should be given velocity goals.
 - Define a service class for startup processes, which are forked by the initialization process, `BPXOINIT`. This service class should be given a velocity goal that is higher than that of other forked children.
- Define classification rules:
 - By default, put forked child processes (subsystem type `OMVS`) into the service class defined for forked children.
 - Put the kernel (with `TRXNAME=OMVS`) into a high-priority Started Task (subsystem type `STC`) service class. Another option is to keep the `OMVS` started procedure in the default started class category, which generally has high priority.
 - Put the initialization process `BPXOINIT` (with `TRXNAME=BPXOINIT`) into a high-priority Started Task (subsystem type `STC`) service class. Another option is to keep the `BPXOINIT` started procedure in the default started class category, which generally has high priority.
 - Startup processes that are forked by the initialization process, `BPXOINIT`, fall under `SUBSYS=OMVS`. These processes are identified by `USERID=OMVSKERN`. Put them in a separate service class as defined above.
 - Other forked child processes (under subsystem type `OMVS`) can be assigned to different service classes based on `USERID`, `ACCTINFO`, or `TRXNAME`.
 - Put the DFSMS buffer manager `SYSBMAS` (with `TRXNAME=SYSBMAS`) into a high-priority Started Task (subsystem type `STC`) service class. Another option is to allow the `SYSBMAS` started procedure to remain in the default started class category which generally has high priority.

Defining service classes for kernel work

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. You can set duration values using the service-units-per-second value reported in the RMF Monitor I Workload Activity report.

The following is a sample service class for forked children. Change these values as appropriate for your installation.

* 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

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.

* Service Class OMVSKERN - OMVS startup processes

Base goal:

#	Duration	Imp	Goal description
1		1	Execution velocity of 40

Your installation may have other special classes of users. If so, you may want to define other service classes for kernel work.

If you have used the PRIORITYGOAL statement in the BPXPRMxx parmlib member to enable the nice(), setpriority(), and chpriority() functions, additional service classes for kernel work must be added. See “Enabling nice(), setpriority(), and chpriority() support” on page 340 for details.

Defining classification rules as needed

Specify the classification rules needed to separate daemons (for example, **inetd**) from other forked children.

Example: This example is a sample classification for subsystem type OMVS:

* Subsystem Type OMVS

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	

Forked spaces can be classified by transaction name.

If you do not define any classification rules, OMVS and BPXOINIT will run under the rules for subsystem type STC, which typically is defined to have high priority. If needed, you can define a classification rule for subsystem type STC to ensure that the kernel, the initialization process, BPXOINIT, and the DFSMS buffer manager, SYSBMAS, run as a high-priority started tasks.

Example: In this example, STC1 is a service class for high-priority started tasks:

* 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	
. .	.		STC1	
. .	.		...	

Defining BPXPRMxx parmlib members in IEASYSxx

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

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

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

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

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

OMVS=(F1,LI)

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

OMVS=(F2,LI)

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

OMVS=(F3,L3)

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

Example: In order to have different HFS data sets mounted at /etc on each system in the sysplex:

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

Customizing the BPXPRMxx parmlib members

The BPXPRMxx parmlib member contains the parameters that control processing and the file system.

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

Where is the complete list of BPXPRMxx parmlib members?

z/OS MVS Initialization and Tuning Reference contains a complete description of the BPXPRMxx statements. This section only discusses the BPXPRMxx parmlib members that have planning considerations.

When you complete your installation activities, you have one or two BPXPRMxx members, depending on whether you used ServerPac or CBPDO:

- With ServerPac, you receive two members, as IBM recommends.
- With CBPDO, after you complete all the instructions in the *z/OS Program Directory*, you have the one member that you copied from SYS1.SAMPLIB.

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

Customize these BPXPRMxx members, according to the instructions in this section and the needs of your installation. When customizing, remember to use columns 1 through 71 for data; columns 72 through 80 are ignored.

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

Figure 5 shows the IBM-supplied BPXPRMXX member in SYS1.SAMPLIB for the current release.

```

MAXPROCSYS(900)
  MAXPROCUSER(25)
  MAXUIDS(200)
  MAXFILEPROC(2000)
  MAXPTY(800)
  CTRACE(CTIBPX00)
/*STEPLIBLIST('/etc/steplib') */
/*USERIDALIASTABLE('/etc/tablename') */

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

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

/* FILESYSTYPE TYPE(TFS) */
/*                      ENTRYPOINT(BPXTFS) */

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

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

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

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

FILESYSTYPE TYPE(UDS) ENTRYPOINT(BPXTUINT)
NETWORK DOMAINNAME(AF_UNIX)
          DOMAINNUMBER(1)
          MAXSOCKETS(200)
          TYPE(UDS)

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

/* NETWORK DOMAINNAME(AF_INET6) DOMAINNUMBER(19) */
/*                      /*For IPv6 TYPE(INET) */

/* FILESYSTYPE TYPE(CINET) ENTRYPOINT(BPXCINT) */
/* NETWORK DOMAINNAME(AF_INET) */
/*                      DOMAINNUMBER(2) */
/*                      MAXSOCKETS(2000) */
/*                      TYPE(CINET) */
/*                      INADDRANYPORT(2000) */
/*                      INADDRANYCOUNT(325) */
/* NETWORK DOMAINNAME(AF_INET6) DOMAINNUMBER(19) */
/*                      /*For IPv6 TYPE(CINET)

```

Figure 5. BPXPRMXX parmlib member in SAMPLIB (Part 1 of 2)

```

/* SUBFILESYSTYPE NAME(TCPIP)          */
/*                                TYPE(CINET) */
/*                                ENTRYPOINT(EZBPFINI) */
/*                                DEFAULT      */

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

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

IPCMSGNIDS (500)
IPCMSGQBYTES (2147483647)
IPCMSGQNUM (10000)
IPCSHMNIDS (500)
IPCSHMPAGES (262144)
IPCSHMPAGES (25600)
IPCSHMNSEGS (500)
IPCSEMNIDS (500)
IPCSEMNSEMS (1000)
IPCSEMNOPS (25)
MAXMMAPAREA(40960)

/* MAXFILESIZE(1000) */

MAXCORESIZE(4194304)
MAXASSIZE(209715200)
MAXCPU(1000)
MAXSHAREPAGES(131072)
FORKCOPY(COW)
SYSPLEX(NO)
SUPERUSER(BPXROOT)
TTYGROUP(TTY)
STARTUP_PROC(OMVS)

/* STARTUP_EXEC('Dname(Memname)',SysoutClass) */
/* RUNOPTS('runtime options') */

SYSCALL_COUNTS(NO)
MAXQUEUEDSIGS(1000)
SHRLIBRGNSIZE(67108864)
SHRLIBMAXPAGES(4096)
LIMMSG(NONE)
AUTOCVT(OFF)
RESOLVER_PROC(DEFAULT)
SWA(BELOW)

```

Figure 5. BPXPRMXX parmlib member in SAMPLIB (Part 2 of 2)

You can change some BPXPRMxx values without an IPL:

- The SET™ OMVS and SETOMVS operator commands dynamically change the settings system-wide. “Dynamically changing the BPXPRMxx parameter values” on page 259 indicates which parameter statements can and cannot be dynamically changed.
- The RACF ALTUSER or ADDUSER commands apply settings on a per-user basis for a particular user, such as Lotus Domino®. You can use them for the MAXASSIZE, MAXCPU, MAXFILEPROC, MAXPROCUSER, MAXMMAPAREA, and MAXTHREADS parameters.

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

Defining file systems

The following sections explain how you can customize the FILESYSTYPE, ROOT, MOUNT, NETWORK, and SUBFILESYSTYPE statements to specify your file systems. These statements define the file systems at OMVS initialization.

You can customize the FILESYSTYPE, ROOT, MOUNT, NETWORK, and SUBFILESYSTYPE statements to specify your file systems. For HFS sharing across a sysplex, the SYSPLEX(YES) parameter is required, and you must also specify a value for the VERSION statement. See Chapter 9 for more information.

FILESYSTYPE

The FILESYSTYPE statement defines the type of physical file system.

When you specify SYSPLEX(YES), you must define the filesystype for all systems participating in shared HFS. The easiest way to define FILESYSTYPE is to have a single BPXPRMxx member that contains file system information for each system participating in shared HFS. If, however, you decide to define a BPXPRMxx for each system, the FILESYSTYPE statement must be identical on each system. See “Customizing BPXPRMxx for shared HFS” on page 166 for more information about configuring BPXPRMxx in a sysplex.

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

Table 3 describes some physical file system types (TYPE parameter) and module names (ENTRYPOINT parameter).

Table 3. File system types

File System Type	Description	Module Name
AUTOMNT	Handles automatic mounting and unmounting of file systems.	BPXTAMD
CINET	Handles requests for the AF_INET and AF_INET6 family of sockets. This enables many different AF_INET or dual AF_INET/AF_INET6 physical file systems to be active on the system. See Chapter 21 for information about setting up sockets. If you want to use CINET, you must be using z/OS Communication Services (TCP/IP Services). If you use CINET, you cannot use INET.	BPXTCINT
DFSC	Enables a z/OS user or application running in a Distributed Computing Environment (DCE) to access directories and files in the DFS™ global namespace.	IOECMINI
HFS	Processes file system requests. The HFS statement is necessary if you want to use regular local files.	GFUAINIT
INET	Handles requests for the AF_INET and AF_INET6 family of sockets. You must be using z/OS Communication Services (TCP/IP Services). If you use INET, you cannot use CINET.	EZBPFINI

Table 3. File system types (continued)

File System Type	Description	Module Name
NFS	Handles Network File System requests for access to remote files. For NFS Client you must create a procedure to run a PFS in a colony address space. For more information, see <i>z/OS Network File System Customization and Operation</i> . You can also find information in “Running a physical file system in a colony address space” on page 43.	GFSCINIT
TFS	Handles requests to the temporary file system (TFS).	BPXTFS
UDS	Handles socket requests for the AF_UNIX address family of sockets.	BPXTUINIT
ZFS	Handles Distributed File Service zSeries file system requests.	IOEFSCM

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

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

For more information on VIRTUAL(max) and other FILESYSTYPE PARM("") keywords, see the BPXPRMxx section in *z/OS MVS Initialization and Tuning Reference*.

MOUNT

The MOUNT statement defines the hierarchical file systems to be mounted at initialization and where in the file hierarchy they are to be mounted. All HFS data sets specified on MOUNT statements in the BPXPRMxx parmlib member must be available at IPL time. If an HFS data set is migrated by HSM, then the initialization of OMVS and HSM will deadlock. Neither kernel nor HSM services will be available.

Systems exploiting shared HFS will have I/O to an OMVS couple data set. Because of these I/O operations to the CDS, each mount request requires additional system overhead. You will need to consider the affect that this change will have on your recovery time if a large number of mounts are required on any system participating in shared HFS. For more information on shared HFS, see Chapter 9.

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

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

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

Tip: Use SETOMVS SYNTAXCHECK to check the MVS catalog for the existence of the HFS or zFS data set names listed on each MOUNT statement. This helps ensure that the mounts will succeed.

NETWORK

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

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

```
FILESYSTYPE Type (INET) Entrypoint (EZBPFINI)
NETWORK DOMAINNAME (AF_INET) TYPE(INET)
        DOMAINNUMBER(2) MAXSOCKETS(50000)
NETWORK DOMAINNAME(AF_INET6) DOMAINNUMBER(19) TYPE(INET)
```

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

Tips:

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

ROOT

The ROOT statement defines and mounts the root file system for a hierarchical file system (HFS).

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

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

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

Tip: Use SETOMVS SYNTAXCHECK to check the MVS catalog for the existence of the HFS or zFS data set names listed on each MOUNT statement. This helps ensure that the mounts will succeed.

SUBFILESYSTYPE

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

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

Changes to BPXPRMxx for sockets may also require changes in the user's TCP/IP security system. For more information, see "Setting up TCP/IP security" on page 108. Chapter 21 has details about setting up for sockets.

Defining system limits

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

Table 4 lists the system-wide and process-level limits that can be set in the BPXPRMxx parmlib member. Not all of the statements are explained in this section; see the BPXPRMxx section in *z/OS MVS Initialization and Tuning Reference* for a complete description of each BPXPRMxx statement.

Table 4. System-wide and process-level limits

System-Wide Limits	Process-Level Limits
MAXUIDS	MAXCPU TIME
MAXPTYS	MAXPROCSYS
MAXRTYS	MAXTHREADTASKS
MAXMMAPAREA	MAXFILEPROC
MAXSHAREPAGES	MAXPROCUSER
MAXASSIZE	MAXCORESIZE
IPCMSGNIDS	MAXQUEUEDSIGS
IPCSHMNIDS	MAXTHREADS
IPCSHMSPAGES	MAXASSIZE
IPCSEMNIDS	
IPCMSGQBYTES	
IPCMSGQMNUM	
IPCSHMMPAGES	
IPCSHMNSEGS	
IPCSEMNSEMS	
IPCSEMNOPS	
SHRLIBMAXPAGES	
SHRLIBRGNSIZE	

CTRACE

Use the CTRACE statement to provide tracing while the kernel is starting and to avoid having to issue a TRACE operator command to set tracing options. See for information about specifying your customized component trace parmlib members.

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

LIMMSG

Use the LIMMSG statement to control the display of console messages that indicate when parmlib limits are reaching critical levels. For more information, see "Displaying the status of BPXPRMxx parmlib limits" on page 268.

MAXASSIZE

MAXASSIZE is the maximum region size (in bytes) for an address space that was created by rlogind, telnetd, and other daemons. You can set a system-wide limit in BPXPRMxx and then set higher limits for individual users. Use the RACF ADDUSER or ALTUSER command to specify the ASSIZEMAX limit on a per-user basis as follows:

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

MAXCPU TIME

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

```
ALTUSER userid OMVS(CPU TIME MAX(nnnn))
```

Specifying a MAXCPU TIME or CPU TIME MAX of 86400 seconds disables the JWT timeout the same way that JCL TIME=1440 does.

Tip: MAXCPU TIME specifies the RLIMIT_CPU hard limit resource values that processes receive when they are dubbed a process. RLIMIT_CPU indicates the CPU time that a process is allowed to use, in seconds. The soft limit is obtained from MVS. If the soft limit value from MVS is greater than the MAXCPU TIME value, the hard limit is set to the soft limit. This value is also used when processes are initiated by a daemon process using an exec after setuid(). In this case, both the RLIMIT_CPU hard and soft limit values are set to the MAXCPU TIME value.

- For processes running in or forked from TSO or BATCH, the MAXCPU TIME value has no effect. A superuser can override this value by specifying a new time limit in the spawn inheritance structure on __spawn().
- For processes running in or forked from TSO or BATCH, the MAXCPU TIME value has no effect. The TIME limit is inherited from the parent. If a TIME parameter is specified on the JCL for the started task, then that value is used. If not, then the TIME value is taken from the JES default TIME value.
- For processes created by the rlogind command or other daemons, MAXCPU TIME is the time limit for the address space.

MAXFILEPROC

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

When selecting a value, consider the following factors:

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

Recommendation: Set this value to 256.

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

A process can change the MAXFILEPROC value using the setrlimit() function. Only processes with appropriate privileges can increase their limits.

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))
```

“Dynamically changing certain BPXPRMxx parameter values” on page 259 explains how to dynamically change the MAXFILEPROC value.

MAXMMAPAREA

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

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

Example:

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

The MMAPSIZE parameters allows for a value that is greater than two gigabytes. The SHEMMAX and MEMLIMIT parameters also enable installations to manage the 64-bit space more effectively.

MAXPROCSYS

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

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

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

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

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

“Dynamically changing certain BPXPRMxx parameter values” on page 259 explains how to dynamically change the MAXPROCSYS value.

For an example of MAXPROCSYS settings in BPXPRMxx, see “Tuning process activity” on page 339.

MAXPROCUSER

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

To improve performance, use MAXPROCUSER to limit user activity.

When selecting a value, consider the following factors:

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

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

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

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

Example:

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

For an example of MAXPROCUSER settings in BPXPRMxx, see “Tuning process activity” on page 339.

MAXPTYs

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

Recommendation: Because each user may have more than one session, you should allow four pseudo-TTY pairs for each user (MAXUIDS * 4). Specify a MAXPTYs value that is at least twice the MAXUIDS value.

“Dynamically changing certain BPXPRMxx parameter values” on page 259 explains how to dynamically change the MAXPTYs value. For more information about pseudoterminal files, see “Pseudoterminal files” on page 136.

For an example of MAXPTYs settings in BPXPRMxx, see “Tuning process activity” on page 339.

MAXSOCKETS

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

If you are using AnySockets over SNA, a high value for MAXSOCKETS may use too many resources. You should use a low value instead.

MAXTHREADS

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

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

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

MAXTHREADTASKS

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

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

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

MAXUIDS

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

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

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

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

For an example of MAXUIDS settings in BPXPRMxx, see “Tuning process activity” on page 339.

PRIORITYGOAL

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

If you are using your system to run a critical real-time application program, specify a list of service class names. It is difficult to run both real-time application programs and general users on the same z/OS UNIX system because you cannot restrict any set of users from access to the `nice()` and `setpriority()` functions. For more information, see “Enabling `nice()`, `setpriority()`, and `chpriority()` support” on page 340.

PRIORITYPG

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

If you are using your system to run a critical real-time application program, specify a list of performance group numbers. It is difficult to run both real-time application programs and general users on the same z/OS UNIX system because you cannot restrict any set of users from access to the `nice()` and `setpriority()` functions. For more information, see “Enabling `nice()`, `setpriority()`, and `chpriority()` support” on page 340.

Defining system features

AUTOCVT

If AUTOCVT is set, in order to enable Enhanced ASCII, performance is affected because every read and write operation for a file must be checked to see if conversion is necessary.

Recommendation: Use AUTOCVT(OFF). If you want to enable Enhanced ASCII using another method, see Chapter 13, “Using Enhanced ASCII functionality,” on page 245. That chapter describes Enhanced ASCII in detail.

STEPLIBLIST

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

Step libraries have many uses; one is so that selected users can test new versions of run-time libraries before the new versions are made available to everyone on the system. Customers who do not put the Language Environment Run-Time Library SCEERUN into the linklist should put the SCEERUN data set name in this file.

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

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

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

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

The following is a list of formatting rules for the STEPLIBLIST file that contains the sanctioned list:

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

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

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

Following is a sample sanctioned list file:

```

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

```

You can create or update the sanctioned list file using the `OSTEPLIB` command, which specifies read and execute permissions for all users (permissions 555). The sanctioned list file must be protected from update by nonprivileged users; therefore, only users with superuser authority should be given update access to it.

Because a working copy of the sanctioned list is maintained in storage, an update to the file will take effect when the next `setuid(0)` program is run from a process with read access to the `stepliblist` file.

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

USERIDALIASTABLE

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

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

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

Following is a list of formatting rules for the userid alias table:

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

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

Following is a sample user ID and group name alias table:


```

/*****/
/*                                          */
/*   Name: Sample user ID/group name alias table      */
/*                                          */
/*   Description:  Contains a list of MVS user IDs and their */
/*                  associated alias names.              */
/*                                          */
/*   Alias names may be constructed from the following characters: */
/*                                          */
/*       A B C D E F G H I J K L M N O P Q R S T U V W X Y Z */
/*       a b c d e f g h i j k l m n o p q r s t u v w x y z */
/*       0 1 2 3 4 5 6 7 8 9 . _ - */
/*                                          */
/*       The hyphen shall not be used as the first character. */
/*                                          */
/*****/

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

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

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

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

```

For installation security reasons, you may have to use the user ID alias table. See “Security requirements for ServerPac and CBPDO installation” on page 86 for more information.

The user ID/group name alias table must be protected from update by nonprivileged users; therefore, only users with superuser authority should be given update access to it. All users should be given read access to the file.

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

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

where **/etc/tablename** is the name of your user ID alias table.

AUTHPGMLIST

AUTHPGMLIST specifies the pathname of a hierarchical file system (HFS) file that contains the lists of APF-authorized pathnames and program names. Those lists are called *sanction lists*. For more information about setting up and activating sanction lists, see “Using sanction lists” on page 102.

Example:

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

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

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

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

Customizing other parmlib members

This section describes how to customize these parmlib members:

- ALLOCxx
- COFVLFxx
- CTnBPXxx
- IEADMR00
- SMFPRMxx

ALLOCxx parmlib member to control allocation requests

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

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

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

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

COFVLFxx parmlib member to activate RACF classes

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

CLASS NAME(IRRGMAP)	/* OpenMVS-RACF GMAP table	*/
EMAJ(GMAP)	/* Major name = GMAP	*/
CLASS NAME(IRRUMAP)	/* OpenMVS-RACF UMAP table	*/
EMAJ(UMAP)	/* Major name = UMAP	*/
CLASS NAME(IRRGTS)	/* RACF GTS table	*/
EMAJ(GTS)	/* Major name = GTS	*/
CLASS NAME(IRRACEE)	/* RACF saved ACEEs	*/
EMAJ(ACEE)	/* Major name = ACEE	*/
CLASS NAME(IRRSMAP)	/* Security packet	*/
EMAJ(SMAP)	/* Major name = SMAP	*/

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

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

For information about caching UIDs and GIDs, see “Caching RACF user and group information in VLF” on page 335.

CTnBPXxx parmlib member to control tracing

The CTnBPXxx parmlib member specifies the tracing options for a component trace of z/OS UNIX events:

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

The CTRACE parameter in the BPXPRMxx member specifies the member; see Figure 5 on page 25, where CTIBPX00 is specified.

- One member can be set up to trace all z/OS UNIX events. This member is CTIBPX01. This enables a site to change trace information on the fly to obtain suitable ctrace information for a dump. (CTIBPX00 traces minimum information.)
- Create other members as needed or when requested by the IBM Support Center.

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

Figure 6 shows the IBM-supplied CTIBPX00 member in SYS1.PARMLIB.

```

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

```

Figure 6. CTIBPX00 parmlib member

The statements in CTIBPX00 do the following:

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

You can specify a buffer size between 16KB and 4MB on the BUFSIZE statement. Use the TRACE CT operator command to change the size of the trace buffer.

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

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

Figure 7 shows a member, CTCBPX08, with an OPTIONS statement that requests tracing of events in files and pipes.

```

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

```

Figure 7. Customized CTCBPX08 parmlib member

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

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

IEADMR00 parmlib member to gather dump data

You should change parmlib member IEADMR00 (SYSMDUMP and core dump defaults) to specify:

SDATA=(RGN,SUM,TRT,LPA)

This gathers adequate data without an excessive dump size.

SMFPRMxx parmlib member to specify timeouts

The JWT value in the SMF parmlib member SMFPRMxx specifies how long an idle address space is allowed to wait before it is terminated.

When an address space is dubbed a process, or when a forked or spawned process is created, the process may go into signal-enabled waits. In a signal-enabled wait, the address space is made exempt from long-wait timeouts as specified by the JWT value in the parmlib member SMFPRMxx. This enables parent processes to wait forever while child processes are running. Otherwise, if the parent process is terminated due to job wait timeout, a SIGHUP signal is sent to the running process and work is lost.

However, shell users, whether logged on through TSO/E and the OMVS command, or via rlogin or telnet, are exempt from job wait timeout because the shell is in a signal-enabled wait while waiting for a command from the user. To have shell users be timed out and logged off, you need to specify the TMOUT environment variable in **/etc/profile**. The TMOUT environment variable contains the number of seconds before user input times out. If user input is not received, the shell ends.

If a shell started by the TSO/E OMVS command times out, then the TSO address becomes enabled for job wait timeout processing. This means that if you have JWT=30 (30 minutes) and you have TMOUT=600 (10 minutes), then TSO users who leave their terminals in the shell will time out and be logged off in about 40 minutes.

Customizing /etc

You must copy the files listed in Table 5 to the specified directory.

Table 5. Copying files from /samples to /etc

Used during	Copied	
	From:	To:
System Initialization	/samples/rc	/etc/rc
	/samples/init.options	/etc/init.options

Customizing the OMVS cataloged procedure to run the kernel initialization program

The OMVS cataloged procedure runs a program that initializes the kernel. The STARTUP_PROC statement in the BPXPRMxx parmlib member specifies the OMVS cataloged procedure. The default name is OMVS.

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

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

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

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

- The replacement started procedure must also be a single jobstep procedure that invokes the BPXINIT program (EXEC PGM=BPXINIT). If it invokes any other program, OMVS initialization will fail.
- You need to change the procedure name in the RACF started procedures table or change the definition in the STARTED Class. See “Preparing RACF” on page 56.

Running a physical file system in a colony address space

Physical file systems are sometimes initialized in an address space called a *colony address space*. You can think of these address spaces as extensions of the kernel address space. The NFS Client and DFS Client physical file systems must be set up in a colony address space because they need to use socket sessions to talk to their remote servers and this cannot be done from the kernel. You can choose to set up the TFS in a colony address space also; to make that decision see “Running a temporary file system in a colony address space” on page 45.

Some physical file systems cannot be initialized in colonies; for example, the INET or CINET sockets file systems and HFS.

How to start colonies

To set up a physical file system in a colony address space, create a cataloged procedure in SYS1.PROCLIB to start the colony address space.

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

Example: An NFS Client with the cataloged procedure NFSCLNT is associated with the following FILESYSTYPE statement:

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

The procedure must contain the statement:

```
EXEC PGM=BPXVCLNY
```

For the complete sample NFS client cataloged procedure, see *z/OS Network File System Customization and Operation*.

How to start colonies outside of JES

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

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

where:

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

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

Example:

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

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

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

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

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

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

SYSIN	For standard input.
SYSPRINT	For standard output. If SYSPRINT does not exist, Language Environment looks for SYSTERM or SYSERR. If one of those exists, it will be used. But Language Environment does not dynamically allocate either SYSTERM or SYSERR.
SYSOUT	For standard error. It is also the default message file DD.
CEEDUMP	For capturing dumps formatted by Language Environment

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

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

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

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

For information about setting up security for the colony address space, see Step 4 on page 59.

Running a temporary file system in a colony address space

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

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

Steps for creating a cataloged procedure for a temporary file system

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

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

Example: For cataloged procedure XXXXXX, the FILESYSTYPE statement would be:

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

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

```
EXEC PGM=BPXVCLNY
```

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

Tip: A temporary file system uses private storage for the file system in memory. If you run it in the kernel, then you might run out of virtual storage. However, by starting multiple temporary file systems in colonies, you can create many temporary files or very large temporary files (about 1.5 gigabytes per temporary file system colony).

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

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

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

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

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

For more information on translation into Japanese, refer to Chapter 11.

Although the user can invoke these TSO/E commands from a TSO/E command line, most users invoke TSO/E commands or programs from an ISPF menu. Therefore, you should add these TSO/E commands to an ISPF selection panel. Figure 8 on page 47 shows the ISR@PRIM (ISPF Primary Option Menu) modified to include these commands.

Be sure that the symbol at the start of this statement (2F in Figure 8 on page 47) matches the number specified in the list of options.

5. Add a statement to the list of options for the ISPF shell. Be sure to include a selection number with the statement. In Figure 8 on page 47, this is:

```
% ISH - ISPF shell
```

6. Add a statement to the)PROC section of the panel to invoke the ISPF shell environment. In Figure 8 on page 47, this is:

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

Be sure that the symbol at the start of this statement (ISH in Figure 8 on page 47) matches the number specified in the list of options.

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

See *z/OS ISPF Dialog Developer's Guide and Reference* for information about modifying ISPF selection panels.

Setting up the REXX parameter modules

Among other things, the function call syscalls ('ON') ensures that the SYSCALL host command environment is available in your REXX environment. If the call detects that SYSCALL is not available in your environment, it dynamically adds it.

Performance characteristics for dynamically added host commands are not as good as for host commands that are included in the initial environment. The difference is primarily because every time a command is directed to the SYSCALL host command environment, the TSO/E REXX support loads the module for the SYSCALL host command.

To avoid this, include the SYSCALL host command in the three default TSO/E environments:

Module Name	SYS1.SAMPLIB Member Name	REXX Environment
IRXPARMS	IRXREXX1	MVS
IRXTSPRM	IRXREXX2	TSO
IRXISPRM	IRXREXX3	ISPF

Customizing IRXISPRM provides dramatic performance improvement for the ISPF shell. If your users use REXX programs that use syscall commands from TSO/E or MVS batch, similar performance improvements are possible.

Steps for adding the syscall command to the default environment

Perform the following steps to add the syscall command to the default environment.

1. Locate the label SUBCOMTB_TOTAL and add 1 to its value.

Example: Change:

```
SUBCOMTB_TOTAL DC F'14'
```

to

```
SUBCOMTB_TOTAL DC F'15'
```

2. Locate the label SUBCOMTB_USED and add 1 to its value.

Example: Change:

```
SUBCOMTB_USED DC F'14'
```

to

```
SUBCOMTB_USED DC F'15'
```

3. Locate the end of the subcommand table. This will be just prior to the label PACKTB or PACKTB_HEADER. Add the following lines:

```
SUBCOMTB_NAME_REXXIX DC CL8'SYSCALL '  
SUBCOMTB_ROUTINE_REXXIX DC CL8'BPXWREXX'  
SUBCOMTB_TOKEN_REXXIX DC CL16' '
```

The ISPF dialogs will work without this modification, but performance will be severely affected.

Note: In the SUBCOMTB_TOKEN_REXXIX definition, there are 16 blanks.

4. Assemble and link-edit the module and replace the default TSO/E module. These are normally installed in SYS1.LPALIB.
-

When you are done, you have added the syscall command to the host environment. For more information on customizing the default environments, refer to *z/OS TSO/E REXX Reference*.

Checking for setup errors

After you complete the customization process, you might want to run the Setup Verification Program (SVP) to check for potential setup errors.

Check the z/OS UNIX web site for the program:

<http://www.ibm.com/s390/zos/bpxa1svp.html>

Chapter 4. Using msys for Setup for z/OS UNIX

Overview

You can use the z/OS Managed System Infrastructure for Setup (msys for Setup) to customize z/OS UNIX. With its management directory or repository of configuration data, msys for Setup can significantly reduce the complexity of setting up the z/OS UNIX environment by:

- Guiding you in establishing the basic definitions and values that the HFS and TFS file systems use in the z/OS UNIX environment.
- Assisting you in setting limits on z/OS UNIX system resources.

This chapter provides conceptual and introductory information about msys for Setup for z/OS UNIX. For more specific information about customization, see the help documentation provided with msys for Setup for z/OS UNIX. For details about msys for Setup, see *z/OS Managed System Infrastructure for Setup User's Guide*.

Who should use msys for Setup?

msys for Setup is intended for the system programmer who is setting up a z/OS UNIX configuration for the first time. However, msys for Setup can also be used to collect and update the configuration data from an existing z/OS UNIX configuration. In both cases, msys for Setup for z/OS UNIX is intended to be the vehicle to make additions or modifications to the z/OS UNIX configuration data and to the z/OS UNIX settings themselves.

- For the installation and setting up a z/OS UNIX configuration for the first time, msys for Setup can set limits on system resources and prepare file systems for z/OS UNIX.
- For existing z/OS UNIX installations, msys for Setup provides an efficient method of maintaining and updating z/OS UNIX configuration information.

In both cases, once msys for Setup has been used to customize and activate a z/OS UNIX configuration, you should use the customization panels to update or maintain the configuration as new resources are added.

What is a z/OS UNIX customization task?

A z/OS UNIX customization task includes a series of panels that ask you for specific information about your intended or existing configuration. The panels are self-explanatory and contain extensive help information for completing each panel.

- If you are defining a z/OS UNIX configuration for the first time, the customization panels allow you to specify basic information about your z/OS UNIX environment.
- If you have an existing z/OS UNIX configuration, the customization panels contain information about your configuration, which you can then add to or modify.

The customization task saves all new and updated information from the panels in the management directory. The msys for Setup Update task applies the changes to your host system and updates the information on the current system settings in the management directory. To activate the changes, you can access the msys for Setup log, which contains specific instructions (such as operator commands that you must issue) on how to make the changes effective throughout z/OS UNIX.

If you do not perform the Update task after customizing panel values, msys for Setup saves your values but does not update the information on the current settings in the management directory. If you begin the customization task again, you might overlay values that you previously entered. Or, if you perform the Refresh task, the customization panels will be loaded with the current system settings that resulted from the last time you performed Update (and not from the previous customization for which you did not perform Update).

If you perform Refresh for an existing z/OS UNIX configuration, the panels will contain information about your current configuration.

You can customize this information during Customization, making modifications to your data that you can then apply to the host system and write back to the current system settings in the management directory by performing Update.

After you complete all the tasks for each msys for Setup z/OS UNIX customization task, you will have defined the BPXPRMxx parmlib member.

If you want to change these z/OS UNIX environment values later, you will have to rerun the customization task and update the configuration.

Preparing to use msys for Setup

Requirements: Before you run the z/OS UNIX customization dialogs under msys, you must install z/OS using the ServerPac delivery method. The customization dialogs work only with the release they are shipped with and any subsequent interim releases until the next level of the dialogs is shipped. They do not work with back-level releases.

- For the ServerPac full replacement installation, preliminary z/OS UNIX setup is done. At a minimum, the target system will have:
 - Three hierarchical file systems (HFS file systems): one for root, one for **/etc**, and one for msys.
 - Two BPXPRMxx members as shipped by IBM.
 - An OMVS statement in IEASYS that points to one or more BPXPRMxx members.
 - Minimal security definitions (for example, for key z/OS UNIX cataloged procedures such as OMVS and BPXOINIT).
- At the end of a ServerPac software upgrade installation, the target system will carry forward the system control files that were in use with the previous software level. For instance, they will retain their BPXPRMxx member and whatever customization that was previously done. However, if you did not set up z/OS UNIX in the previous release, you will need to do so after installing z/OS.

Using msys for Setup for z/OS UNIX customization

Recommendation: msys for Setup provides many z/OS UNIX settings based on best practices and current experience. You should use these default settings. If you decide to change any of the defaults, msys for Setup provides help information to assist you. Use the help as an informational guide when adjusting any of the system resources or file systems settings.

Restrictions: The following restrictions apply when using msys for Setup:

- You cannot automount file systems.

- If you want to add an HFS and are creating a new mount point, there is no interface for running the **mkdir**, **chmod**, and **chown** commands on the target system.
- You cannot create a proc in SYS1.PROCLIB.
- You cannot use the RACF plug-in to create a user ID, to test whether there is a STARTED class profile for OMVS and pull information from it, or to create a new STARTED class profile.

Chapter 5. Establishing UNIX security

Overview

To provide data and system security, the security administrator and security auditor need to set up and maintain security with the tasks that are described in this chapter.

z/OS UNIX provides security mechanisms that work with the security offered by the z/OS system. A security product is required, either RACF or an equivalent security product. This chapter assumes that you are using RACF. If you are using an equivalent security product, you should refer to that product's documentation. If you do not have a security product, you must write SAF exits to simulate all of the functions.

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

Additionally, IBM's multilevel security functions for z/OS build on the work done on MVS to meet the B1 criteria. Using multilevel security, an installation can provide a high level of security on a z/OS system. However, IBM does not claim that its multilevel security functions comply with any particular criteria, and IBM has not sought any formal certification of the functions. For more information on multilevel security refer to *z/OS Planning for Multilevel Security*.

Use the following as reference documents:

- *z/OS Security Server RACF Security Administrator's Guide*
- *z/OS Security Server RACF System Programmer's Guide*
- *z/OS Security Server RACF Command Language Reference*

In this chapter

This chapter covers the following subtasks.

Subtasks	Associated procedure (see . . .)
Managing group identifiers and user identifiers (GIDs and UIDs)	"Steps for obtaining security information about users" on page 67 "Steps for setting up field-level access for the OMVS segment of a user profile" on page 68
Allowing all z/OS UNIX users to transfer file ownership to any UID or GID on the system	"Steps for assigning selected users to transfer ownership of any file" on page 77 "Steps for setting up the CHOWN.UNRESTRICTED profile" on page 78
Giving user superuser authority	"Steps for setting up BPX.SUPERUSER" on page 79
Changing superusers from UID(0) to a unique nonzero UID	"Steps for changing a superuser from UID(0) to a unique nonzero UID" on page 80
Defining RACF groups to z/OS UNIX groups	"Steps for creating z/OS UNIX groups" on page 73

Subtasks	Associated procedure (see . . .)
Setting up the FILE.GROUPOWNER.SETGID profile	"Steps for setting up the FILE.GROUPOWNER.SETGID profile" on page 90
Setting up sanction list processing	"Steps for creating a sanction list" on page 104 "Steps for activating the sanction list" on page 105
Maintaining the security level of the system	"Steps for maintaining the security level of the system" on page 107

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

- "Comparing UNIX security and z/OS UNIX security" on page 290.
- "Establishing the correct level of security for daemons" on page 291.

Preparing for RACF

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

Other security topics include:

- Chapter 17, for rlogin security considerations
- Chapter 18, for information about preparing security for servers
- "Steps for preparing the security program for daemons" on page 294

Preparing RACF

A SAMPLIB member, BPXISEC1, is provided with z/OS UNIX. This sample TSO/E CLIST provides all the RACF commands needed for the security setup discussed throughout this document. Use this sample member to set up your security environment.

This installation assumes that you have installed z/OS. To prepare RACF for z/OS UNIX, the security administrator should take the following steps:

1. The OMVS cataloged procedure runs a program that initializes the kernel. To define the user ID and group ID specified for OMVS, enter the RACF commands as shown in the following example:

```

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

```

 - When you create the RACF user ID for OMVSKERN, you should use the NOPASSWORD option to create it as a protected user ID. Protected user IDs can neither be used to log on to the system nor be revoked by incorrect password attempts.
 - Specify the RACF name for the group: OMVSGRP in the example. Because the processes created by **/usr/sbin/init** inherit the GID of the BPXOINIT, do not permit the OMVSGRP to any MVS resources, unless programs you start using **/etc/rc** need to be permitted to these resources. For more information, see "Customizing /etc/rc" on page 201.

In this example, the GID is 1. However, OMVSGRP may have any group ID.

- The TSO/E segment is not needed because NOPASSWORD will prevent the OMVSKERN user ID from being used with TSO/E. This prevents a user logon from interfering with the OMVSKERN user ID.
- Assign UID(0) to the kernel user ID (OMVSKERN). Any programs forked by **/etc/rc** receive their authority from the user ID assigned to the BPXOINIT process. Use the same user ID for BPXOINIT as you assigned to the kernel (OMVS). The BPXOINIT process and any programs forked by the kernel's descendants have superuser authority.
- Specify the home directory for the kernel: the root (/).
- To define the default shell for processes run with the OMVSKERN user ID, specify:

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

Example:

```
export _BPX_ACCOUNT=AccountingData
```

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

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

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

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

- Add the following entries to ICHRIN03.

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

- If OMVS is not a trusted procedure, add OMVS without making it trusted, using one of the following methods. (See step 3 for additional measures needed if the kernel is not trusted.)

- Add it to the RACF STARTED class:

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

If you add any other entries after this, issue
SETROPTS RACLIST(STARTED) REFRESH

They will be picked up on the next START.

- Add it to ICHRIN03, as shown in the following example:

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

- Add BPXOINIT without making it trusted, using either one of these methods:

- Add it to the RACF STARTED class:

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

- Add it to ICHRIN03:

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

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

Either fulfill the three following conditions:

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

For example,

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

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

Or:

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

For example,

```
ADDUSER SMORG
ADDSD ('SMORG.HFS') UACC(NONE) OWNER(SMORG)
PERMIT 'SMORG.HFS' ACCESS(UPDATE) ID(OMVSKERN)
```

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

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

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

- Add it to the RACF STARTED class:

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

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

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

Managing group identifiers and user identifiers (GIDs and UIDs)

This section describes using RACF with z/OS UNIX. It describes factors to consider when using RACF to manage z/OS UNIX group identifiers (GIDs) and z/OS UNIX user identifiers (UIDs).

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

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

For complete information on auditing in the RACF environment, see *z/OS Security Server RACF Auditor's Guide*.

Setting up users and groups

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

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

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

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

Activating supplemental groups

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

Restrictions:

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

For more information about list-of-groups checking, see *z/OS Security Server RACF Security Administrator's Guide*.

Defining z/OS UNIX users to RACF

You can define users as described in this section. Alternatively, you can use the ISPF shell to set up existing users with unique UIDs.

Rules:

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

To define a user, do the following:

1. Log on to the TSO/E user ID with RACF SPECIAL authority.
2. Authorize a TSO/E user to use z/OS UNIX by entering:
 - A RACF ADDUSER command for each new user to be given access to z/OS UNIX. The ADDUSER command creates a RACF user profile.
 - A RACF ALTUSER command for each current TSO/E user who is to be given access to z/OS UNIX. The ALTUSER command changes a current RACF user profile.

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

3. Assign a home directory for each user through the HOME subparameter on the ADDUSER or ALTUSER command. If the home directory is **/u/john**, specify:

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

Recommendation: The home directory should be fully qualified (**/u/john**). If a home directory is partially specified (for example, **john**) problems can arise during process initialization. Then create that home directory for each user. The home directory, like all file names, is case-sensitive. It is recommended that the username in the home directory be entered in lowercase.

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

If the home directory is the root, specify:

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

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

- If a user accesses the shell from TSO/E, the user ID is folded to uppercase
 - With rlogin, the user ID is case sensitive. If the alias table (USERIDALIASTABLE) is not set up, then case does not matter and the user ID is folded. If the alias table is being used and the user ID is found in it, then the case-sensitive user ID for UNIX activity is used.
4. Specify an initial program for each user through the PROGRAM subparameter of the ADDUSER or ALTUSER command. You should specify:

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


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

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

5. Do one of the following to connect a user to an already-defined RACF group. The RACF group must have an OMVS GID for the user to access z/OS UNIX resources.
 - Specify the RACF group on the DFLTGRP parameter on the RACF ADDUSER command. The specified group becomes the user's default group. Do not specify a RACF group on the RACF ADDUSER command. Your current connect group becomes the user's default group.
 - Enter a RACF CONNECT command to connect a user to the RACF group. Specify the DFLTGRP parameter on the RACF ALTUSER command to change the user's default group to the RACF group with an OMVS GID.

To use z/OS UNIX, the default group of the user must have a GID defined.
6. z/OS UNIX performs SYSOUT tailoring for every forked address space. When defining the users, code the WORKATTR parameter to specify the user's name and address. The name and address appear on the user's SYSOUT output.

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

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

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

The DFLTGRP parameter places user ID JOHN in the RACF group ENGNGP7, which has an OMVS GID of 678 (see “Creating z/OS UNIX groups” on page 73). The OMVS parameter on the ADDUSER command does the following:

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

On an open system, a working directory is normally defined in lowercase letters and usually has the user's TSO/E user ID as its name—for example, **/u/john**. If a REXX exec or CLIST extracts the user ID with a *&userid* variable, the value returned is in uppercase: JOHN. If the REXX exec or CLIST appends the returned value to **/u**, the result is **/u/JOHN**. **/u/john** and **/u/JOHN** are two different directories. You should consider this behavior in using REXX execs, CLISTs, C programs, or programs using the callable services where the functions return user IDs.

- Specifying the WORKATTR for user ID JOHN allows daemons to create processes with the correct accounting and SYSOUT defaults. For example, if

JOHN logs into the system using a rlogin command from a workstation, a new process will be created for JOHN using the attributes from the WORKATTR.

Setting up default OMVS segments

This section explains how the installation can create a default OMVS segment for users and groups that do not have an OMVS segment defined. Batch jobs submitted on the local system are assigned a user ID and group. If the assigned user ID or group has an OMVS segment defined, it will be used. If the assigned user ID or group does not have an OMVS segment defined, a default OMVS segment may be used, as explained below. Batch jobs submitted from remote systems may not be assigned a user ID and group by the local system. In this case, they must have a user ID and group explicitly coded. If they do not, the default OMVS segment will not be used and the job will fail with an error message.

An installation may want to set up default OMVS segments for these reasons:

- Some users need to use sockets and do not need any other UNIX services. In the past, they could open sockets without any other special permissions.
If users run z/OS UNIX applications other than socket applications such as FTP, then it is strongly recommended that those users be given OMVS segments. Otherwise, the resources created by these applications on behalf of these users will be available to other users running with the same default OMVS segment.
To prevent the potential misuse of the default OMVS segment, the callable services kill(), pidaffinity(), trace(), and sigqueue() are not supported when running with the default OMVS segment.
- Some users just want to experiment with the shell and do not have an OMVS segment defined.
Be aware that some experimenting users will have access to files belonging to other experimenting users because all these files will be owned by the same UID/GID.

The default OMVS segments will reside in a USER profile and a GROUP profile. The names of these profiles are selected by the installation, using a profile in the FACILITY class. The name of the FACILITY class profile is BPX.DEFAULT.USER. The application data field in this profile will contain the user ID, or the user ID/group ID, of the default profiles.

If you define BPX.DEFAULT.USER, then all users will be able to access z/OS UNIX. If you want to prevent certain users from being able to access z/OS UNIX services, you can define an OMVS segment with no UID. Then the job will fail when those users attempt to use a UNIX service. If users must be allowed (for example, for FTP or other socket use) but you do not want them to use the shell, consider defining the initial program for the default user as **/bin/echo**. Then users with the default UID will not be able to use the shell.

Steps for setting up default OMVS segments

Before you begin: You need to know what you will put in the OMVS segment.

- **UID.** You might want to make the UID different enough so that it stands out when used. You can either make it very high or very low, but do not set the UID of the default user to 0.
- **HOME.** There are several options when defining the home directory for the default user.
 - Define a directory just like any other user. This directory would then be used concurrently by many users that do not have an OMVS segment. This is not recommended.

- Define the HOME directory as the root (/). The users will not have write access, but should not need to update their home directory.
- Define the HOME directory in the **/tmp** directory.
- **PROGRAM.** Define the default shell in this field. If you do not want users running in a shell environment with the default UID and GID, then define the PROGRAM parameter as:
PROGRAM('/bin/echo')

This will cause any attempt to enter the shell to terminate.

In addition to UID, HOME, and PROGRAM, the user limits values (such as CPUTIMEMAX and PROCUSERMAX) from the default OMVS segment are used. If you expect to have a lot of users running with the default segment, you might want to set the user limits higher than the system limits to accommodate them. See *z/OS Security Server RACF Command Language Reference* for information about the user limit values.

Perform the following steps to set up a default OMVS segment.

1. Define a group ID to the system that will be used as an anchor for the default OMVS group segment.

Example:

```
ADDGROUP OMVSLTG OMVS(GID(777777))
```

When defining the default group, the main consideration is what you put in the OMVS segment. You might want to make this GID sufficiently different so that it stands out when used. You can either make it very high or very low.

-
2. Define a user ID to the system that will be used as an anchor for the default OMVS user segment.

Example: To define a default user:

```
ADDUSER OMVSLTU DFLTGRP(OMVSLTG) NAME('OMVS DEFAULT USER')
NOPASSWORD OMVS(UID(999999) HOME('/u/omvsflt'))
PROGRAM('/bin/sh'))
```

You can use the NOPASSWORD option with the ADDUSER command for OMVSLTU to indicate that it is a protected user ID and cannot be used to log on to the system.

-
3. Create BPX.DEFAULT.USER and put the name of a default user ID (or user ID/group ID) in the application data field of that profile. The assumption is that the FACILITY class has already been activated. The USER profile of the default user ID and the GROUP profile of the default group ID must exist, and must contain OMVS segments containing a UID and GID, respectively

If you try to use the default segment, RACF will check to make sure that it exists.

Example: The following example shows the specification of the default user ID and group ID which provide the anchor for the default OMVS segment for users and groups.

```
RDEFINE FACILITY BPX.DEFAULT.USER APPLDATA('OMVSLTU/OMVSLTG')
SETROPTS RACLIST(FACILITY) REFRESH
```

To set up a default for the USER OMVS segment only, the format is:

```
RDEFINE FACILITY BPX.DEFAULT.USER APPLDATA('OMVSLTU')
SETROPTS RACLIST(FACILITY) REFRESH
```

You cannot set up a default GROUP OMVS segment alone. If you take the time to set up individual OMVS segments for all users, it is likely that you would want to set up OMVS segments for groups because there are usually far fewer groups than users defined to RACF.

When you are done, you have set up the default OMVS segment for your installation.

Note: BPX.DEFAULT.USER is used as follows:

- A user requests a kernel service; the kernel then dubs the user.
- The kernel calls the security product to extract the UID, GID, HOME, and PROGRAM information.
- The security product attempts to extract the OMVS segment associated with the user. If it is not defined, it tries to extract and use the OMVS segment for the default user that was listed in the BPX.DEFAULT.USER profile.
- A similar process is done to obtain a GID when the user's default group does not have an OMVS segment.

Supporting case-sensitive user IDs

XPG4 compliance requires the operating system to support case-sensitive user IDs that can optionally contain periods, dashes, and underscores. To provide this capability, the installation can define a user ID alias table.

This user ID alias support allows an XPG4-compliant program to work correctly with a user ID that exploits user ID naming conventions not normally tolerated on z/OS. However, this support stops at the boundary between XPG4-defined functions and the rest of z/OS.

All security checks done by traditional z/OS services are based on the z/OS user ID. You can only log on to TSO/E using a valid z/OS user ID.

Recommendation: There are many ways in which use of a non-standard user ID conflicts with the running of normal business workloads. It is therefore strongly recommended that installations not define a user ID alias table. If you still believe that it is in your installation's best interest to exploit case-sensitive user IDs, see "USERIDALIASTABLE" on page 37.

Security implications of programs running in the HFS

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

Programs in unauthorized libraries can be further divided into system-controlled libraries, which are protected from general user modification, and libraries that are not system-controlled. Libraries that are not system controlled are not considered safe for anyone to run. This code is generally a local version of a program that the

owner has created or modified. Users with special privileges, must use caution when running such programs. If a programmer with RACF SPECIAL or authority to update APF-authorized libraries runs a program from an uncontrolled library, it is possible for the program to take advantage of the caller's authority to compromise the integrity of the system.

Prior to OS/390® V2R6, **dbx** could not be used on programs running in an APF-authorized address space. The BPX.DEBUG resource in the FACILITY class enables you to debug APF-authorized programs, using ptrace via **dbx**. For more information about BPX.DEBUG, see “Setting up the BPX.* FACILITY class profiles” on page 83.

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

The entire HFS is considered to be an unauthorized library. You can authorize individual programs within the HFS as APF-authorized by setting the APF-extended attribute. HFS programs that are APF-authorized behave the same as other programs that are loaded from APF-authorized libraries. If a program running in an APF-authorized address space attempts to load a program from the 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. This is consistent with the way that Contents Supervisor rejects requests to LINK, LOAD, or ATTACH unauthorized programs from an authorized environment.

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

1. You can link-edit the program into an APF-authorized library and turn on the sticky bit in the HFS, using the **chmod** command. (For an explanation of the sticky bit, see *z/OS UNIX System Services User's Guide*.)
2. You can use the **extattr** command to set the APF-authorized extended attribute of the file.

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

Authority checks

The system uses two types of user and group IDs to check a user's authority to access different resources:

- **MVS data sets:** The system uses:
 - The TSO/E user ID in the RACF user profile
 - The RACF group name for the user's current group
 - The RACF group name for each group the user is connected to, if list-of-group checking is active
- **local files:** The system uses:
 - The effective UID
 - The effective GID
 - The GIDs for the supplemental groups, if list-of-group checking is active

Users need a UID and GID defined when entering the TSO/E OMVS command and for certain kernel services. If they do not have an OMVS segment defined, then an

attempt is made to access the default OMVS segment (see “Defining z/OS UNIX users to RACF” on page 60). If the user does not have a UID or GID defined or if a default is not set up, then the OMVS command or the service fails.

Users also need search authority to all directories in the pathname for their home directory. Set these permissions for each directory using the **chmod** command and either the MODE operand of the TSO/E MKDIR command or the *mode* option of the **mkdir** command that creates a directory. For more information, see “Controlling access to files and directories” on page 89.

Obtaining security information about a group

This section explains how to obtain security information about groups.

Steps for obtaining security information about a group

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

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

1. Log on to your TSO/E user ID.

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

Example: To list the GID information for the RACF group ENGNP7:

```
LISTGRP ENGNP7 OMVS NORACF
```

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

Obtaining security information about users

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

Steps for obtaining security information about users

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

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

1. Log on to your TSO/E user ID.

2. Issue a RACF LISTUSER command with the OMVS operand and the user's TSO/E user ID.

Example: To list the OMVS information for TSO/E user ID JOHN:

```
LISTUSER JOHN OMVS NORACF
```

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

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

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

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

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

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

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

Example:

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

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

Example: The following example shows commands for the three fields.

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

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

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

-
3. Activate the FIELD class with the RACF SETROPTS command.

Example:

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

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

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

Defining group identifiers (GIDs)

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

Although the same GID can be assigned to multiple RACF groups, it is not recommended. If you assign the same GID to multiple groups, control at an individual group level is lost because the GID is used in z/OS UNIX security checks. RACF groups that have the same GID assignment are treated as a single group during z/OS UNIX security checks. They will need to use the SHARED keyword of the RACF ADDGROUP or ALTGROUP command if SHARED.IDS is implemented. For more information about SHARED.IDS, see *z/OS Security Server RACF Security Administrator's Guide*.

If you are using NFS, see “Assigning UIDs and GIDs in an NFS network” on page 71 for more information.

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

Defining user identifiers (UIDs)

Assigning UIDs to single users

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

Rule: 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.

If you are using NFS, see “Assigning UIDs and GIDs in an NFS network” on page 71 for more information.

Assigning UIDs to multiple users

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

However, you 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.

Rule: If SHARED.IDS is implemented, you will need to use the SHARED keyword of the RACF ADDUSER or ALTUSER command.

Setting user limits

You can control the amount of resources consumed by certain z/OS UNIX users by setting individual limits for these users. The resource limits for the majority of z/OS UNIX users are specified in the BPXPRMxx parmlib member. These limits apply to all users except those with UID(0), which indicates superuser authority. Rather than assigning superuser authority to application servers and other users so they can exceed BPXPRMxx limits, you can individually set higher limits to these users. Setting user limits allows you to minimize the number of assignments of superuser authority at your installation and reduces your security risk.

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

CPUTIMEMAX	Maximum CPU time (RLIMIT_CPU)
ASSIZEMAX	Maximum address space size (RLIMIT_AS)
FILEPROCMAx	Maximum number of files per process
PROCUSERMAX	Maximum number of processes for this UID
THREADSMAX	Maximum number of threads per process
MMAPAREAMAX	Maximum memory map size

After you set individual user limits for users who require higher resource limits, you should consider removing their superuser authority, if they have any. You should also reevaluate your installation's BPXPRMxx limits and consider reducing these limits. See "Customizing the BPXPRMxx parmlib members" on page 24 for more information.

Defining protected user IDs

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

Defining the terminal group name

Certain shell commands, such as **mesg**, **talk**, and **write** require pseudoterminals to have a group name of *TTY*. When a user logs in, or issues the OMVS command from TSO/E, the group name associated with these terminals is changed to *TTY*. As part of installation, you had to define the group *TTY* or use the group alias support as described in "Security requirements for ServerPac and CBPDO installation" on page 86.

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

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

Managing user and group assignments

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

- Use AUTOGID or AUTOUID to have UIDs and GIDs automatically assigned to the user. This is the recommended method.
- Use a RACF database unload utility to move RACF data into a DATABASE 2 (DB2®) database and then use the Structured Query Language (SQL) to query the database.
- Use the ISPF shell to perform the tasks of defining users and groups.

Assigning UIDs and GIDs in an NFS network

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

Assigning identifiers for users

As in similar open systems, the same number can be used for the GID.

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

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

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

To assign a non-unique UID, you can use the SHARED keyword of the RACF ADDGROUP or ALTGROUP command if SHARED.IDS is implemented.

Assigning identifiers for groups

As in similar open systems, the same number can be used for the GID. All groups should be assigned unique GIDs. If you assign groups the same GIDs, you should warn users of the following effects:

- The groups share ownership and access to the same files.

- Security audit records show the GID, but do not show the RACF group if it was in the supplemental group list; see “Activating supplemental groups” on page 60.
- Services such as the `getgrgid()` function cannot distinguish which group is meant. The services return data about one of the groups, but which group is unpredictable.

To assign a non-unique GID, you can use the `SHARED` keyword of the RACF `ADDGROUP` or `ALTGROUP` command if `SHARED.IDS` is implemented.

Upper limits for GIDs and UIDs

RACF allows for UIDs and GIDs within the range of 0–2 147 483 647. However, the **pax** and **tar** utilities cannot handle values above 16 777 216. If you use **pax** or **tar** to copy files with a UID or GID above 16 777 216, UIDs or GIDs may be incorrectly assigned to the restored files. (The details of this limitation is described in “Limitations of pax and tar.”) Because they are commonly used utilities, you should take this problem into consideration before assigning UIDs or GIDs above 16 777 216.

Limitations of pax and tar

When using **pax** or **tar** to transport files, UIDs/GIDs greater than 16 777 216 will be incorrectly restored unless the USTAR format is used and the user/group name associated with the UID/GID exists on the target system.

This happens because **pax** and **tar** uses one of two archive formats: the original tar format or the USTAR format which is an enhanced version of the original format. The original tar format provides two 8-character fields to store the UID and GID. The USTAR format provides these same fields plus two additional 32-character fields to hold the user and group name associated with the UID and GID. USTAR is the default format for **pax**. The default format for **tar** is the original **tar** format, but the `-U` option can be used to cause **tar** to use the USTAR format.

The architected standard for either format only provides 8 octal characters to represent a UID or GID. Consequently, the largest UID/GID that can be represented is 16 777 216 (octal “77777777”). UIDs/GIDs greater than this are stored in the archive using the eight high-order octal characters which results in the incorrect UID/GID being stored.

For example, assume that a file has a UID of 2147483647 (the maximum value allowed by RACF). In octal, this is represented as 17777777777. In this case, **pax** and **tar** would store the first 8 characters “17777777” as the UID. In decimal, this is 4194303. So, UIDs of 2 147 483 647 are incorrectly stored as 4 194 303.

When restoring a UID/GID, if the USTAR format was used during dumping, **pax** and **tar** will first attempt to set the UID/GID using the user/group name stored in the archive. (Of course, the user must have the appropriate privileges to set the UID/GID). If this name is defined on the target system, then the UID/GID is set to whatever UID/GID is associated with the name defined on the target system. (The UID/GID is set, whether or not it matches the UID/GID in the archive, which means that this could be a problem if the name stored on the target system is coincidental rather than intentional). If the name is not defined on the system, or if the archive is using the original tar format, then the UID/GID stored in the archive is used. In this case, if the UID/GID was originally greater than 16 777 216, then the incorrect UID/GID is restored.

In summary, UIDs and GIDs greater than 16 777 216 might not be correctly restored by **pax** and **tar**. Using the USTAR format can avoid this, but only if the target system has the same user or group name defined.

Creating z/OS UNIX groups

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

Steps for creating z/OS UNIX groups

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

1. Log on to the TSO/E user ID with RACF SPECIAL authority.
2. Issue one of the following commands. Base your choice on your particular situation.

If you want to . . .	Then issue. . .
Define a new RACF group profile and have it be used as a z/OS UNIX group	The ADDGROUP command. Example: To define a RACF group profile named SYS1 and to give it a GID of 575, issue: <pre>ADDGROUP OMVSGRP SUPGROUP(SYS1) OWNER(SYS1) OMVS(GID(575))</pre> Result: You have defined a RACF group profile and created a z/OS UNIX group.
Change a current RACF group profile and have it used as a z/OS UNIX group	The ALTGROUP command. Example: To add a GID of 678 to the current RACF group ENGNGP7, issue: <pre>ALTGROUP ENGNGP7 OMVS(GID(678))</pre> Result: You have created a z/OS UNIX group. Tips: • You can use AUTOGID to automatically assign an unused GID. Example: <pre>ALTGROUP ENGNGP7 OMVS(AUTOGID)</pre> • You can use the ISPF shell to assign OMVS GIDs to all groups.

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

When you are done, you have created a z/OS UNIX group. When the user connects to the system (for example, logs on to a TSO/E session), one group is

selected as the user's current group. When a user becomes a z/OS UNIX user, the GID of the user's current group becomes the effective GID of the user's process. The user can access resources available to members of the user's effective GID.

Assigning superuser attributes

Your installation defines certain system programmers, users, and started procedures as superusers, who can change the contents of any file, install products, manage processes, and perform other administrative activities. When not doing activities that require superuser authority, that person or program joins the majority of users or programs with user authority, which permits access to their own files and certain files to which they have access, according to permission bits.

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

What can superusers do?

Superusers can do the following:

- 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.

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

Defining superusers with appropriate privileges

As you are defining users, you might want to define some of them with appropriate superuser privileges. This chapter describes the three ways of assigning superuser authority.

- Using the UNIXPRIV class profiles, the preferred way. See “Using UNIXPRIV class profiles” on page 75.
- Using the BPX.SUPERUSER resource in the FACILITY class. See “Using the BPX.SUPERUSER resource in the FACILITY class” on page 79.
- Assigning a UID of 0, which is the least desirable way. See “Assigning a UID of 0” on page 83.

For specific installation requirements regarding superuser authority, see “Security requirements for ServerPac and CBPDO installation” on page 86.

While some functions require a UID of 0, in most cases you can choose among the three ways. When choosing among them, try to minimize the number of “human” user IDs (as opposed to started procedures) set up with UID(0) superuser authority. 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. And, the UNIXPRIV class allows

you to do other privileged functions, such as mounting a file system. Both these definitions are similar to having UID(0) in that, before RACF grants access to a system resource or use of it, the system checks these definitions.

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

Using UNIXPRIV class profiles

You can define profiles in the UNIXPRIV class to grant RACF authorization for certain z/OS UNIX privileges. By defining profiles in the UNIXPRIV class, you can specifically grant certain superuser privileges with a high degree of granularity to users who do not have superuser authority. This 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.

Table 6 shows each resource name available in the UNIXPRIV class, the z/OS UNIX privilege associated with each resource, and the level of access required to grant the privilege.

Table 6. Resource names in the UNIXPRIV class for z/OS UNIX privileges

Resource name	z/OS UNIX privilege	Minimum access required
CHOWN.UNRESTRICTED ¹	Allows all users to use the chown command to transfer ownership of their own files.	None required
FILE.GROUPOWNER.SETGID	Specifies that a directory's set-gid bit is used to determine the group owner of any new objects created within the directory.	None required
RESTRICTED.FILESYS.ACCESS	Specifies that RESTRICTED users cannot gain file access by virtue of the 'other ' permission bits.	None required
	Can be overridden for a specific user/group.	READ
SHARED.IDS	Allows users to assign UID and GID values that are not unique.	READ
SUPERUSER.FILESYS.ACLOVERRIDE	Specifies that ACL contents override the access that was granted by SUPERUSER.FILESYS.	None required
	Can be overridden for a specific user/group.	User/group must have the same access that would be required to SUPERUSER.FILESYS while accessing the file

Table 6. Resource names in the UNIXPRIV class for z/OS UNIX privileges (continued)

Resource name	z/OS UNIX privilege	Minimum access required
SUPERUSER.FILESYS ²	Allows user to read any local file, and to read or search any local directory.	READ
	Allows user to write to any local file, and includes privileges of READ access.	UPDATE
	Allows user to write to any local directory, and includes privileges of UPDATE access.	CONTROL (or higher)
SUPERUSER.FILESYS.CHANGEPERMS	Allows users to use the chmod command to change the permission bits of any file and to use the setfacl command to manage access control lists for any file.	READ
SUPERUSER.FILESYS.CHOWN	Allows user to use the chown command to change ownership of any file.	READ
SUPERUSER.FILESYS.MOUNT	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 option. Users permitted to this profile can use the chmount shell command to change the mount attributes of a specified file system.	READ
	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.	UPDATE
SUPERUSER.FILESYS.QUIESCE	Allows user to issue quiesce and unquiesce commands for a file system mounted with the nosetuid option.	READ
	Allows user to issue quiesce and unquiesce commands for a file system mounted with the setuid option.	UPDATE
SUPERUSER.FILESYS.PFSCTL	Allows user to use the pfsctl() callable service.	READ
SUPERUSER.FILESYS.VREGISTER ³	Allows a server to use the vreg() callable service to register as a VFS file server.	READ
SUPERUSER.IPC.RMID	Allows user to issue the ipcrm command to release IPC resources.	READ
SUPERUSER.PROCESS.GETPSENT	Allows user to use the w_getpsent() callable service to receive data for 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.	READ

Table 6. Resource names in the UNIXPRIV class for z/OS UNIX privileges (continued)

Resource name	z/OS UNIX privilege	Minimum access required
SUPERUSER.PROCESS.KILL	Allows user to use the kill() callable service to send signals to any process.	READ
SUPERUSER.PROCESS.PTRACE ⁴	Allows user to use the ptrace() function through the dbx debugger to trace any process.	READ
SUPERUSER.SETPRIORITY	Allows user to increase own priority.	READ
Notes: 1. See “Using the CHOWN.UNRESTRICTED profile” on page 78. 2. Authorization to the SUPERUSER.FILESYS resource provides privileges to access only local files. No authorization to access Network File System (NFS) files is provided by access to this resource. 3. The SUPERUSER.FILESYS.VREGISTER resource only lets a server like NFS initialize. Users that are connected as clients through facilities such as NFS do not get special privileges based on this resource or other resources in the UNIXPRIV class. 4. Authorization to the BPX.DEBUG resource is also required to trace processes that run with APF authority or BPX.SERVER authority.		

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

Assigning superuser privileges

The example in “Steps for assigning selected users to transfer ownership of any file” applies to the superuser privileges shown in Table 6 on page 75, except the privilege associated with the CHOWN.UNRESTRICTED resource (see “Using the CHOWN.UNRESTRICTED profile” on page 78).

Steps for assigning selected users to transfer ownership of any file

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

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

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

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

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

2. Assign selected users or groups as appropriate.

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

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


```
SETOPTS CLASSACT(UNIXPRIV)
```

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

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

```
SETOPTS RACLIST(UNIXPRIV)
```

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

```
SETOPTS RACLIST(UNIXPRIV) REFRESH
```

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

Allowing z/OS UNIX users to change file ownerships

On z/OS UNIX systems, superusers can change the ownership of any file to any UID or GID on the system. General users can only change the ownership of files that they own, and only to one of their own associated GIDs. You can assign all z/OS UNIX users to transfer ownership of files they own to any UID or GID on the system, or you can assign selected users to transfer ownership of any file to any UID or GID.

To allow all z/OS UNIX users to transfer ownership of files they own to any UID or GID on the system, create a discrete profile in the UNIXPRIV class called CHOWN.UNRESTRICTED. If this profile is defined on your system, all z/OS UNIX users can issue the **chown** command to transfer ownership of files that they own.

Rule: CHOWN.UNRESTRICTED must be a discrete profile. Matching generic profiles are ignored. Access lists are not needed for this profile because RACF will only check that CHOWN.UNRESTRICTED exists.

Using the CHOWN.UNRESTRICTED profile

Steps for setting up the CHOWN.UNRESTRICTED profile:

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

1. Define the CHOWN.UNRESTRICTED profile in the UNIXPRIV class.

```
RDEFINE UNIXPRIV CHOWN.UNRESTRICTED
```

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

```
SETOPTS CLASSACT(UNIXPRIV)
```

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

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

```
SETROPTS RACLIST(UNIXPRIV)
```

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

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

When you are done, you have set up the CHOWN.UNRESTRICTED profile in the UNIXPRIV class. z/OS UNIX users can now transfer ownership for files that they own.

Using the SUPERUSER.FILESYS.CHOWN profile

To allow selected z/OS UNIX users to transfer ownership of any file to any UID or GID, create a profile in the UNIXPRIV class protecting a resource called SUPERUSER.FILESYS.CHOWN. See “Assigning superuser privileges” on page 77 for an example of authorizing users using the SUPERUSER.FILESYS.CHOWN resource.

Using the BPX.SUPERUSER resource in the FACILITY class

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

Steps for setting up BPX.SUPERUSER

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

Perform the following steps to set up BPX.SUPERUSER.

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

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

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

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

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

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

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

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

Tips:

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

Example: To add the user ID SYSPROG to the SUPERUSR group:

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

Then permit this group to BPX.SUPERUSER.

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

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

Deleting superuser authority

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

Example: To remove superuser authority from user ID JOHN:

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

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

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

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

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

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

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

If you choose this method . . .	Then . . .
Have RACF automatically assign an unused UID.	1. Delete the UID from the user's OMVS segment. Example: ALTUSER JOHN OMVS(NOUID) 2. Issue the ALTUSER command with the AUTOUID keyword. Example: ALTUSER JOHN OMVS(AUTOUID) Result: Message IRR52177I identifies the new UID.
Use the ISPF shell to assign the next available UID.	1. Delete the UID from the user's OMVS segment. Example: ALTUSER JOHN OMVS(NOUID) 2. Assign a new UID, using the ISPF shell. Tip: You can display the user's OMVS segment to see the UID that was assigned by the ISHELL. Example: LISTUSER JOHN OMVS
Manually assign the UID. If the installation manually manages the UIDs assigned to users, select the next available UID and assign it to the user. Tip: To make sure the UID you selected is not already in use by another user, issue: SEARCH CLASS(USER) UID(7)	Use the ALTUSER command. Example: Assume that the next available UID is 7 and the user ID is JOHN. To reassign the UID, issue: ALTUSER JOHN OMVS(UID(7))

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

Example: For user ID John:

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

Tip: You may choose to RACLIST the FACILITY class afterwards. This step is optional. If you do so, then you will have to do a REFRESH after the user ID is permitted to the FACILITY class.

Example:

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

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

Example: The home directory is `/u/john`. Issue the following command to update the ownership of the files.

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

Result: The **chown** command changes the owning UID of the **/u/john** directory to 7 and changes the owning UID of all files and subdirectories of the **/u/john** directory.

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

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

Switching in and out of superuser authority

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

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

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

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

Tip: When running in this manner, editing a file with the OEDIT command (OEDIT with PF6) returns you to the TSO/E address space where your original authority is still in place.

- Enter the ISPF shell using the ISHELL command or a dialog selection. From the ISPF shell, you can select the option to switch to superuser state. You can then manage the file system using ISPF shell functions while in the superuser state.

If you enter the ISPF shell, switch to superuser and then exit the ISPF shell, you may lose superuser authority. If the ISPF shell is the only process in the address space, you will lose all connection to kernel services when the ISPF shell terminates. If there is another dubbed process in this address space (for example, another ISPF shell, or a local shell), it will share the UID with the ISPF shell process. For example, you can open an ISPF shell on both sides of a split screen. When you toggle to superuser in one ISPF shell, it affects the address space and therefore, both ISPF shells are now superuser. Regardless of which ISPF shell terminates first, the address space retains its UIDs until the ISPF shell is used to toggle back, or the last process is undubbed.

- Enter the shell using rlogin or telnet. Once in the shell, enter the **su** command to create a nested shell that runs with superuser authority.
- After gaining superuser authority in the ISPF shell, you can split the screen in ISPF and enter the OMVS command. The shell that is started inherits the superuser authority set up in the ISPF shell. Note that for privileged shells (when the effective UID does not match the real UID, or the effective GID does not match the real GID) **\$HOME/.profile** is not run. If the file **/etc/suid_profile** exists, it will be run.
- If you are permitted to the BPX.SUPERUSER resource, then you can get superuser access through REXX.
- Use the **su** command from BPXBATCH. Run a job using BPXBATCH following one of these examples that shows a copy of the file:
 - On the **PARM=** statement, do the following:

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

This pipes the result of the **echo** command (that is, the **copy** command) into the **su** command.

- With `PARM='SH su'`, code:
`//STDIN DD PATH '/yourpath/input.stuff',PATHOPTS=(ORDONLY)`
- With no parameters coded at all, create a file that has the **su** command in it.
`//BATBPX1 EXEC PGM=BPXBATCH`
`//STDIN DD PATH='/yourpath/suinput.stuff',PATHOPTS=(ORDONLY)`

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

Also, when you set up your own **\$HOME/.profile** as superuser, specify the **/usr/sbin** directory in your `PATH` environment variable because certain superuser utilities are in that directory instead of the **/bin** directory, such as **automount**. For more information about the profile, see “Customizing **\$HOME/.profile**” on page 197.

Assigning a UID of 0

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

Recommendation: If you want to assign a UID of 0, also assign a secondary user ID with a nonzero UID for activities other than system management. For example, you would assign:

User ID	SMORG	UID(0)	- used for system maintenance
User ID	SMORG1	UID(7)	- used for regular programming

Example: In the following example, the `ALTUSER` command gives the TSO/E user ID `SMORG` superuser authority, makes the root directory the home directory, and causes invocation of the shell in response to a TSO/E OMVS command. If the shell is to be installed, specify the `HOME` and `PROGRAM` parameters; if a shell is not to be installed, omit the `HOME` parameter. This user must be in a RACF group, usually `SYS1`, and the group must have an OMVS GID associated with it.

```
ALTUSER SMORG OMVS(UID(0) HOME('/') PROGRAM('/bin/sh'))
ALTGROUP SYS1 OMVS(GID(0))
```

You might choose to assign `UID(0)` to multiple RACF user IDs. However, try to minimize the use of `UID(0)` by using the two methods previously described. Assignment of `UID(0)` should be limited to the user associated with started procedures such as the OMVS kernel and the user who installs the ServerPac. It should be avoided for the user IDs belonging to the real users whenever possible.

Tip: If `SHARED.IDS` is implemented, you may need to use the `SHARED` keyword because `UID(0)` is likely to be used by several IDs. For example:

```
ALTUSER SMORG OMVS(UID(0) SHARED HOME('/') PROGRAM('/bin/sh'))
```

Setting up the BPX.* FACILITY class profiles

For security reasons, you may need to define these FACILITY class profiles:

- **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()

Daemon authority is required only when a program does a `setuid()`, `seteuid()`, `setreuid()`, or `spawn()` user ID to change the current UID without first having issued a `__passwd()` call to the target user ID. In order to change the MVS identity without knowing the target user ID's password, the caller of these services must be a superuser. Additionally, if a BPX.DAEMON FACILITY class profile is defined and the FACILITY class is active, the caller must be permitted to use this profile. If a program comes from a controlled library and knows the target UID's password, it can change the UID without having daemon authority.

For more information about BPX.DAEMON, see "Establishing the correct level of security for daemons" on page 291.

- **BPX.DAEMON.HFCTL**

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 BPX.DEBUG can use `ptrace` (via **dbx**) to debug programs that run with APF authority or with BPX.SERVER authority.

- **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. See for more information.

- **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. *z/OS Security Server RACF Security Administrator's Guide* has more information about BPX.NEXT.USER.

- **BPX.FILEATTR.PROGCTL**

Controls which users are allowed to set the program control attribute in an HFS file. Programs marked with this attribute can execute in server address spaces that run with a high level of authority. See "Step for defining programs in HFS files to program control" on page 297 for more information.

- **BPX.FILEATTR.SHARELIB**

Indicates that extra privilege is required when setting the shared library extended attribute via the `chattr()` callable service. This prevents the shared library region from being misused. See “Defining HFS files as shared library programs” on page 299 for more information.

- **BPX.JOBNAME**

Controls which users are allowed to set their own job names by using the `_BPX_JOBNAME` environment variable or the inheritance structure on spawn. Users with READ or higher permissions to this profile can define their own job names.

- **BPX.SAFFASTPATH**

Enables faster security checks for file system and IPC constructs. For more information, see “Enabling the fastpath support for system authorization facility (SAF)” on page 281.

- **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()`

For more information about BPX.SERVER, see Chapter 18 and “Establishing the correct level of security for daemons” on page 291.

- **BPX.SMF**

Checks if the caller attempting to cut an SMF record is allowed to write an SMF record. It also tests 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**

Allows users to switch to superuser authority. For more information about BPX.SUPERUSER, see “Assigning superuser attributes” on page 74.

- **BPX.WLMSEVER**

Controls access to the WLM server functions `_server_init()` and `_server_pwu()`. It also controls access to these C language WLM interfaces:

- `QuerySchEnv()`
- `CheckSchEnv()`
- `DisconnectServer()`
- `DeleteWorkUnit()`
- `JoinWorkUnit()`
- `LeaveWorkUnit()`
- `ConnectWorkMgr()`
- `CreateWorkUnit()`
- `ContinueWorkUnit()`

A server application with read permission to this FACILITY class profile can use the server functions, as well as the WLM C language functions, to create and manage work requests.

Security requirements for ServerPac and CBPDO installation

Before you can do the ServerPac or CBPDO installation, or install maintenance, you need to satisfy certain security requirements.

1. The user ID must be UID=0 or permitted to the BPX.SUPERUSER resource in the RACF FACILITY class, and be connected to a group that has a GID.
2. The user ID must be permitted read access to the BPX.FILEATTR.APF and BPX.FILEATTR.PROGCTL FACILITY classes (or BPX.FILEATTR.* if you choose to use a generic name for both resources).

Example: To define BPX.FILEATTR.APF and BPX.FILEATTR.PROGCTL, issue:

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

These commands are also provided in SYS1.SAMPLIB.

```
PERMIT BPX.FILEATTR.APF CLASS(FACILITY) ID(your_userid) ACCESS(READ)
PERMIT BPX.FILEATTR.PROGCTL CLASS(FACILITY) ID(your_userid) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

Or, if you choose to use a generic facility:

```
SETROPTS GENERIC(FACILITY)
RDEFINE FACILITY BPX.FILEATTR.* UACC(NONE)
SETROPTS CLASSACT(FACILITY)
SETROPTS RACLIST(FACILITY)

PERMIT BPX.FILEATTR.* CLASS(FACILITY) ID(your_userid) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

3. Define the following user ID and group IDs in your security data base. Even though they are lowercase in the example, these names should be defined in uppercase for ease of use and manageability.

Group IDs

uucpg

TTY

User IDs

uucp

Rules:

- a. **The GID and UID values assigned to these IDs cannot be used by any other IDs. They must be unique.** If you assign the same GID to multiple groups, control at an individual group level is lost, because the GID is used in z/OS UNIX security checks. Because RACF groups that have the same GID assignment are treated as a single group during the z/OS UNIX security checks, the sharing of resources between groups might happen unintentionally. Likewise, the sharing of UIDs allows each user access to all of the resources associated with the other users of that shared UID. The shared access includes not only z/OS UNIX resources such as files, but also includes the possibility that one user could access z/OS UNIX resources of the other user that are normally considered to be outside the scope of z/OS UNIX.
- b. **You must duplicate the required user ID and group names in each security database, including the same UID and GID values in the OMVS segment.** This makes it easier to transport the HFS data sets from test systems to production systems. For example, the group name *TTY* on System 1 must have the same GID value on System 2 and System 3. If it is not possible to synchronize your databases you will need to continue running the FOMISCHO job against each system after z/OS UNIX is installed.

The following sections describe how to define these IDs to RACF. (If you are using an equivalent security product, refer to that product's documentation.) All the RACF commands are issued by a TSO/E user ID with RACF SPECIAL authority. Three procedures are described:

- “If you use uppercase group and user IDs”
- “If you use mixed-case group and user IDs” on page 88
- “If you have problems with names such as UUCP, UUCPG, and TTY” on page 88

If you use uppercase group and user IDs

If you use only uppercase group and user IDs on your system, RACF users can use the BPX1SEC1 sample in SAMPLIB. They can also use the ADDGROUP or ADDUSER commands to define the group IDs and user IDs, as shown in the following examples.

Examples:

1. To define the TTY group:

```
ADDGROUP TTY (OMVS(GID(2)))
```

where 2 is an example of a unique group ID on your system. Do not connect users to this group. This is the same group that is specified on the TTYGROUP statement in the BPXPRMxx parmlib member on your target system.

2. To define the UUCPG group:

```
ADDGROUP UUCPG OMVS(GID(8765))
```

where 8765 is an example of a unique group ID on your system.

3. To define the UUCP user ID, issue:

```
ADDUSER UUCP DFLTGRP(UUCPG) PASSWORD(xxxxxxx)  
OMVS(UID(396) HOME('/usr/spool/uucppublic') PROGRAM('/bin/sh'))
```

where 123456 is an example of a unique account number and 396 is an example of a unique OMVS UID. Do not use UID(0).

If you use mixed-case group and user IDs

If you already use mixed-case group and user IDs on your system and the user (*uucp*) and group (*uucpg*) do not conflict with existing names, perform the steps for uppercase IDs in “If you use uppercase group and user IDs” on page 87.

It is not necessary to add the lowercase or mixed-case names to your alias table, mapping them to uppercase. Using the alias table degrades performance and increases systems management and complexity. When lowercase or mixed-case names are not found in the alias table, or there is no table active, they are folded to uppercase. For more information about the alias table, see “USERIDALIASTABLE” on page 37.

If you have problems with names such as UUCP, UUCPG, and TTY

Examples: If names such as *uucp*, *uucpg*, and *TTY* are not allowed on your system (or if they conflict with existing names), the following examples show the RACF commands to define the group ID and user IDs.

1. To define a group ID instead of the *TTY* group, issue:

```
ADDGROUP XXTTY OMVS(GID)2))
```

where 2 is an example of a unique group ID on your system, and *XXTTY* is replaced by a 1-to 8-character group ID of your choice. Do not connect users to this group. This would be the same group name to be specified in the *TTYGROUP* statement in the *BPXPRMxx* parmlib member on your target system.

2. To define a group ID instead of the *UUCPG* group, issue:

```
ADDGROUP xxuucpg OMVS(GID(8765))
```

where 8765 is an example of a unique group ID on your system, and *xxuucpg* is replaced by a 1-to 8-character group ID of your choice.

3. To define a *uucp* user ID,

```
ADDUSER xxuucp DFLTGRP(UUCPG) PASSWORD(xxxxxxx)  
OMVS(UID(396) HOME('/usr/spool/uucppublic') PROGRAM('/bin/sh'))
```

where 396 is an example of a unique UID (do not use a UID of 0) and *xxuucp* is replaced by a user ID of your choice. This is a normal user ID which owns all the *UUCP* files and directories. Use this user ID when editing configuration files or performing other administrative tasks.

4. Set up a user ID alias table.

Tip: Using the alias table causes poorer performance and increases systems management costs and complexity. For more information about the alias table, see “USERIDALIASTABLE” on page 37.

If you do not have a user ID alias table defined, you will need to create one. This must be done first on your driving system and then on any system image using this product. This fits in with the IBM strategy to place all customized data in the */etc* directory. This table is specified by the *USERIDALIASTABLE* keyword in the *BPXPRMxx* parmlib member. Because the user ID name alias table must be protected from update by nonprivileged users, only users with superuser authority should be given update access to it. All users should have read access to the file.

Your user ID alias table will need to contain your MVS chosen names and the associated required names. Your chosen MVS user ID and group names must be located in columns 1-8 and the associated aliases must be located on the same line in columns 10-17.

```

:groups
XTTY          TTY
XXUUCPG      uucpg
:userids
XXUUCP       uucp

```

5. Activate the user ID alias table. If you are already using the user ID alias table, new database queries will yield the new alias if the userid performing the query has read/execute access to the userid/group name alias table. The table is checked every 15 minutes and refreshed if it has been changed. If a change needs to be activated sooner, you can use the SETOMVS or SET OMVS operator commands.

If you are not using the user ID alias table, you can use the SET OMVS operator command to activate it now. For example:

```
SET OMVS USERIDALIASTABLE=/etc/tablename
```

where **/etc/tablename** is the name of your user ID alias table. You can also use the SETOMVS operator command.

6. Update your BPXPRMxx parmlib member, specifying the USERIDALIASTABLE to make this change permanent for your next IPL.
7. Perform these tasks on all of your driving, test, and production system images.

For more information, see:

- “Defining z/OS UNIX users to RACF” on page 60
- “Creating z/OS UNIX groups” on page 73
- *z/OS MVS System Commands*
- *z/OS Security Server RACF Security Administrator’s Guide*
- *z/OS Security Server RACF Command Language Reference*

Defining cataloged procedures to RACF

If a cataloged procedure starts a program that uses z/OS UNIX or its resources, the procedure should be defined to RACF. An example is the Resource Measurement Facility (RMF) Monitor III Gatherer (RMFGAT).

The RMFGAT started task must be associated with a user ID using ICHRINO3 or the STARTED class, and the user ID that you assign to it must be defined to RACF and needs to have a UID. The user ID must also belong to a group that has a GID. You can use the user ID RMFGAT, but it can be any RACF-defined user ID.

Example: The following example gives RMFGAT a UID of 123 and designates the root directory as its home directory:

```
ADDUSER RMFGAT DFLTGRP(OMVSGRP) OMVS(UID(123) HOME('/')) NOPASSWORD
```

If you have a server address space like RMF, which needs a UID to call UNIX services but does not require a specific UID, you can define default UIDs and GIDs. For more information, see “Setting up default OMVS segments” on page 63.

Controlling access to files and directories

The system provides security for local files by verifying that a z/OS UNIX user can access a directory, a file, and every directory in the path to the file.

The system does a security check for a file, FIFO special file (named pipe), character special file, and directory. It does not check an unnamed pipe, because

this pipe can be accessed only by the parent process that created the pipe and by child processes of the creating process. When the last process using an unnamed pipe closes it, the pipe vanishes.

Every file and directory has security information, which consists of:

- File access permissions (including an ACL, if one exists)
- UID and GID of the file
- Audit options that the file owner can control
- Audit options that the security auditor can control

The file access permission bits that accompany each file provide discretionary access control (DAC). These bits determine the type of access a user has to a file or directory.

The following sections assume that ACLs are not being used. Go to “Using access control lists (ACLs)” on page 95 for more information about ACLs.

Setting classes for a user’s process

The access permission bits are set for three classes. When a user’s process accesses a file, the system determines the class of the process and then uses the permission bits for that class to determine if the process can access the file. For a file, a process can be in only one class. The class for a process can be different for each file or directory.

The class is one of the following:

- **Owner class:** Any process with an effective UID that matches the UID of the file.
- **Group class:** Any process with an effective GID or supplemental group GID that matches the GID of the file when the UIDs do not match.
- **Other class:** Any process that is not in the owner or group class, such as when neither the UIDs or GIDs do not match.

By default, the system sets the UID and GID of the file when the file is created:

- The UID is set to the effective UID of the creating process.
- The GID is set to the GID of the owning directory. You can define FILE.GROUPOWNER.SETGID to change this behavior; see “Using the FILE.GROUPOWNER.SETGID profile.”

To change the UID of a file, a person with superuser authority, or the file owner if CHOWN.UNRESTRICTED is defined to the UNIXPRIV class, can enter a **chown** command or use the `chown()` function. To change the GID of a file, a superuser or the file owner (that is, a process in the owner class) can enter a **chgrp** command or use the `chgrp()` function. You can define profiles in the UNIXPRIV class to grant RACF authorization for certain z/OS UNIX privileges, as explained in “Using UNIXPRIV class profiles” on page 75.

Using the FILE.GROUPOWNER.SETGID profile

To specify that, by default, the group owner of a new HFS file is to come from the effective GID of the creating process, you need to set up a profile in the UNIXPRIV class called FILE.GROUPOWNER.SETGID.

Steps for setting up the FILE.GROUPOWNER.SETGID profile:

Perform the following steps to set up the FILE.GROUPOWNER.SETGID profile.

1. Define the FILE.GROUPOWNER.SETGID profile.

RDEFINE UNIXPRIV FILE.GROUPOWNER.SETGID

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

SETOPTS CLASSACT(UNIXPRIV)

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

SETOPTS RACLIST(UNIXPRIV)

If SETOPTS RACLIST processing is already in effect for the UNIXPRIV class, you must refresh SETOPTS RACLIST processing in order for the FILE.GROUPOWNER.SETGID profile to take effect.

SETOPTS RACLIST(UNIXPRIV) REFRESH

When you are done, you have set up the FILE.GROUPOWNER.SETGID profile. The set-gid bit for a directory determines how the group owner is initialized for new objects created within the directory.

- If the set-gid bit is on, then the owning GID is set to that of the directory.
- If the set-gid bit is off, then the owning GID is set to the effective GID of the process.

Tip: When a new file system is mounted, you must turn on the set-gid bit of its root directory if you want new objects within the file system to have their group owner set to that of the parent directory.

Accessing files

Table 7 shows types of access and the permissions granted by the accesses.

Table 7. File access types and permission bits

Access	Permission for file	Permission for directory
Read	Permission to read or print the contents.	Permission to read, but not search, the contents.
Write	Permission to change, add to, or delete from the contents.	Permission to change, add, or delete directory entries.
Execute or Search	Permission to run the file. This permission is used for executable files.	Permission to search the directory.

To access local files, users need:

- Read and search permission to all directories in the pathnames of files the user should use. Read permission is required for some options of some commands.
- Write permission to all directories in which the user will be creating files or directories.
- Read permission, write permission, or read and write permission, as appropriate to all files that the user needs to access.
- Execute permission to executable files that the user needs to run.

For information about file access permissions, see *z/OS UNIX System Services User's Guide*.

Changing the permission bits for a file

To change the permission bits for a file, use one of the following:

- The ISPF shell
- The **chmod** command. You can use it to change individual bits without affecting the other bits. You can also use the **setfacl** command to change permission bits (see “Managing ACLs” on page 96).
- The `chmod()` function in a program. The function changes all permission bits to the values in the *mode* argument.

The file owner or a superuser can use the **chmod** command or `chmod()` function, or you can define a profile in the UNIXPRIV class to grant RACF authorization. The file mode creation mask does not affect the permission value that was specified by either **chmod** or `chmod()`.

Changing the owner or group for a file

An interactive user might need to change the UID or GID for a file. To protect the data in a file from unauthorized users, the system controls who can change the file access:

- To change the owner and, optionally, the group of a named file, the superuser can enter a **chown** command. The new owner can be identified with a user ID or a UID. The group, if specified, can be identified with a RACF group name or a GID.

The CHOWN.UNRESTRICTED profile allows all users to use the **chown** command to transfer ownership of their own files.

SUPERUSER.FILESYS.CHOWN allows users to use **chown** to change ownership of any file.

- To change the group of a named file to a specified GID, the superuser or the file owner can enter a **chgrp** command. The new group can be identified with a TSO/E group ID or a GID.

Creating a set-user-ID or set-group-ID executable file

A superuser or the file owner can use a **chmod** command or `chmod()` function to change two options for an executable file. The options are set in two file mode bits:

- Set-user-ID (S_ISUID) with the `setuid` option
- Set-group-ID (S_ISGID) with the `setgid` option

If one or both of these bits are *on*, the effective UID, effective GID, or both, plus the saved UID, saved GID, or both, for the process running the program are changed to the owning UID, GID, or both, for the file. This change temporarily gives the process running the program access to data the file owner or group can access.

In a new file, both bits are set *off*. Also, if the owning UID or GID of a file is changed or if the file is written in, the bits are turned *off*.

In shell scripts, these bits are ignored.

Protecting data

Local files and directories are protected by RACF security rules. You can use permission bits to control access; access control lists (ACLs) can also be used in conjunction with permission bits. For more information, see “Using access control lists (ACLs)” on page 95.

Permission bit information is stored in the file security packet (FSP) within each file and directory. (ACLs may also be stored with the file.) Permission bits allow you to specify read authority, write authority, or search authority for a directory. They also allow specification of read, write, or execute authority for a file. Because there are three sets of bits, separate authorities can be specified for the owner of the file or directory, the owning group, and everyone else (like RACF’s universal access authority, or UACC). The owner is represented by a UID. The owning group is represented by a GID. Access checking compares the user’s UID and GID to the ones stored in the FSP.

When a security decision is needed, the file system calls RACF and supplies the FSP (and ACL, if one exists). RACF makes the decision, does any auditing, and returns control to the file system. RACF does not provide commands to maintain the FSP (and ACL), but it does provide SAF services that do the FSP (and ACL) maintenance. z/OS UNIX provides commands that invoke these SAF services.

For information about using RACF authorization to grant privileges for use of local files and directories, see Table 6 on page 75.

Obtaining security information for a file

Users with search access to the directories in the pathname and, for some options, read access to the directories can check a file’s security information, including the access permissions. They do not need read access to the file being checked. Programs can also check security information for files.

To check the security information, do one of the following:

- Use the ISPF shell
- Enter the **ls -l** or **ls -E** shell command.
- Run a stat() or fstat() function in a program.

In response, the system displays the TSO/E user ID and the RACF group name that correspond to the file’s UID and GID. The system displays the UID and GID only if it cannot find the corresponding TSO/E user ID and RACF group name.

For **ls -l**, the permission bits appear as 11 characters.

tfffgggoooa

The characters in this format mean:

Character	Meaning
t	Identifies the type of file or directory:
—	Regular file
b	Block-special file (not supported for z/OS UNIX)
c	Character- special file
d	Directory
e	External link
l	Symbolic link
p	FIFO special file
s	Socket file type

Character	Meaning
fff	Owner permissions • First character: Read access • Second character: Write access • Third character: Execute or, for a directory, search
ggg	Group permissions • First character: Read access • Second character: Write access • Third character: Execute or, for a directory, search
ooo	Other permissions • First character: Read access • Second character: Write access • Third character: Execute or, for a directory, search
a	If 'a' is a plus sign, then the file contains extended ACL entries. Use the getfacl command to display the ACL entries.

The permissions **fff**, **ggg**, and **ooo** are displayed as:

Character	Position	Meaning
–	Any	No access
r	First	Read access
w	Second	Write access
x	Third	Execute (or, for a directory, search)
s	Third (owner only)	Execute permission for owner, set-user-ID set
S	Third (owner only)	No execute permission for owner, set-user-ID set
s	Third (group only)	Execute permission for group, set-group-ID set
S	Third (group only)	No execute permission for group, set-group-ID set
t	Third (other only)	Execute permission for other, sticky bit set
T	Third (other only)	No execute permission for other, with sticky bit set

For example, **rwX** means read, write, and execute permission. Permission for a directory is often **r-x**, which means read and search. If a plus sign follows the permissions, then the file contains extended ACL entries. Use the **getfacl** command to display the ACL entries.

If you issue **ls-E**, it displays extended attributes for regular files. An additional four characters follow the original 10 characters:

```
total 11
-rwxr-xr-x+ -ps-    1 ROOT  SYS1  101 Mar 12 19:32 her
-rwxrwxrwx  a-s-    1 ROOT  SYS1  654 Mar 12 19:32 test
-rwxr-xr-x  a---    1 ROOT  SYS1   40 Mar 12 19:32 temp
-rwxr--r--  ap-l    1 ROOT  SYS1  572 Mar 12 19:32 foo
-rwxr--r--  --sl    1 ROOT  SYS1  640 Mar 12 19:33 abc
```

- a** The program runs APF-authorized if linked AC=1.
- p** The program is considered program controlled.
- s** The program is enabled to run in a shared address space.
- l** The program is loaded from the shared library region.
- The extended attribute is not set.

Using access control lists (ACLs)

Use access control lists (ACLs) to control access to files and directories by individual user (UID) and group (GID). ACLs are used in conjunction with permission bits. They are created, modified, and deleted using the **setfacl** shell command. To display them, use the **getfacl** shell command. You can also use the ISHELL interface to define and display ACLs.

The HFS and zFS file systems support ACLs. It is possible that other physical file systems will eventually support z/OS ACLs. Consult your file system documentation to see if ACLs are supported.

Before you can begin using ACLs, you must know what security product is being used. The ACLs are created and checked by RACF, not by the kernel or file system. If a different security product is being used, you must check their documentation to see if ACLs are supported and what rules are used when determining file access.

Notes:

1. The phrases “default ACL” and “model ACL” are used interchangeably throughout z/OS UNIX documentation. Other systems that support ACL have default ACLs that are essentially the same as the directory default ACLs in z/OS UNIX.
2. According to the X/Open UNIX 95 specification, additional access control mechanisms 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.

ACLs and ACL entries

There are three kinds of ACLs:

- *Access ACLs* are ACLs that are used to provide protection for a file system object.
- *File default ACLs* are default ACLs that are inherited by files created within the parent directory. The file inherits the default ACL as its access ACL. Directories also inherit the file default ACL as their file default ACL.
- *Directory default ACLs* are default ACLs that are inherited by subdirectories created within the parent directory. The directory inherits the default ACL as its directory default ACL and as its access ACL.

Inheritance is the act of automatically associating an ACL with a newly created object. Administrative action is not needed. See “Working with default ACLs” on page 98 for more information.

There are two kinds of ACL entries:

- *Base ACL entries* are the same as permission bits (owner, group, other). You can change the permissions using **chmod** or **setfacl**. They are not physically part of the ACL although you can use **setfacl** to change them and **getfacl** to display them.
- *Extended ACL entries* are ACL entries for individual users or groups; like the permission bits, they are stored with the file, not in RACF profiles. Each ACL type (access, file default, directory default) can contain up to 1024 extended ACL entries. Each extended ACL entry specifies a qualifier to indicate whether the entry pertains to a user or a group, the actual UID or GID itself, and the

permissions being granted or denied by this entry. The allowable permissions are read, write, and execute. As with other UNIX commands, **setfacl** allows the use of either names or numbers when referring to users and groups.

Managing ACLs

Rules: You need to be aware of the following rules when managing ACLs for files or directories.

- You must either be the file owner or have superuser authority (UID=0 or READ access to SUPERUSER.FILESYS.CHANGEPERMS in the UNIXPRIV class).
- You must activate the FSSEC class before ACLs can be used in access decisions.

Example: The following RACF command activates the FSSEC class:

```
SETROPTS CLASSACT(FSSEC)
```

You can define ACLs prior to activating the FSSEC class. If you define default ACLs, they can be inherited by new objects when the FSSEC class is inactive. If the FSSEC class is not active, the standard POSIX permission bit checks are done, even if an access ACL exists. You can still display ACL information.

If files are deleted, ACLs are automatically deleted.

Working with access ACLs

The **getfacl** and **setfacl** commands are used to manage ACLs. Following are a few examples to help you get started. For details on these commands, and on other commands that support ACLs, see *z/OS UNIX System Services Command Reference*.

Examples:

1. Permit user Joe and group Admins to the file named **/etc/inetd.conf** with read and write authority.

```
setfacl -m user:joe:rw-,group:admins:rw- /etc/inetd.conf
```

The **-m** option modifies ACL entries, or adds them if they do not exist.

2. Display the ACL that was created in Step 1.

```
getfacl /etc/inetd.conf
#file: /etc/inetd.conf
#owner: BPXROOT
#group: SYS1
user::rw-
group::r--
other::r--
user:JOE:rw-
group:ADMINS:rw-
```

3. Perform the same operation as in Step 1, but at the same time, set the base permission bits to prevent access by anyone other than the file owner.

```
setfacl -s user::rw-,group:---,other:---,user
user:joe:rw-,group:admins:rw- /etc/inetd.conf
```

The **-s** option replaces the contents of an ACL with the entries specified on the command line. It requires that the base permissions be specified. The base permissions are specified similarly to extended ACL entries, except that there is no user or group name qualifier.

4. Delete the ACL that was created in Step 3.

```
setfacl -D a /etc/inetd.conf
```

The **-D a** option specifies that the access ACL is to be deleted. The permission bits remain as specified in Step 3 on page 96. When a file deleted, its ACL is automatically deleted; there is no extra administrative effort required.

5. Take the ACL from **FileA** in the current directory, and apply it to **FileB**, also in the current directory.

```
getfacl FileA | setfacl -S - FileB
```

The shell pipes the output of **getfacl** to the input of **setfacl**. The **-S** option of **setfacl** says to replace the contents of the file's ACL with ACL entries specified within a file, and the "-" is a special case file name designating stdin. Thus, you can maintain a list of ACL entries within a file, and use that file as input to a **setfacl** command. You might use this ability to implement a "named ACL" for a given project, such as in Step 6.

6. The file **/u/joeadmin/Admins** contains a list of ACL entries for users and groups who need to support some administrative work. The file contains ACL entries, one per line, in the format that **setfacl** expects and which **getfacl** displays. These people must be granted access to all of the directories within the file system subtree starting and including **/admin/work**.

```
setfacl -S /u/joeadmin/Admins $(find /admin/work -type d)
```

This example uses shell command substitution to use the output of the **find** command as input to the **setfacl** command. The **/u/joeadmin/Admins** file may for example contain:

```
user::rwx
group:---
other:---
u:user1:rwx
u:user2:rwx
g:group1:rwx
```

7. Give Lucy read and write access to every file within Fred's home directory for which Ricky has read and write access.

```
setfacl -m user:lucy:rw- $(find ~fred -acl_entry user:ricky:rw)
```

You can use the **find** command to search for various ACL criteria. In this example, it is used to find files containing ACL entries for Ricky, in which Ricky has at least read and write access.

Tip: You can use an access ACL on the parent directory to grant search access only to those users and groups who should have file access. The access ACL of the parent directory can have been automatically created as the result of a directory default ACL on its parent. Make sure that the 'other' and perhaps the 'group' search permission bit is off for the parent directory.

Recommendations: When creating ACLs, consider the following:

- To minimize the impact to performance, keep ACLs as small as possible, and permit groups to files instead of individual users. The pathlength of the access check will increase with the size of an ACL, but will be smaller than the associated checking would be for a RACF profile with the same number of entries in its access list.
- Do not disable ACLs after you have used ACLs for a while and have created many entries. Only consider disabling ACLs if you have not used them very long. If you have been using ACLs to grant, rather than deny, access to particular users and groups, then disabling ACLs will likely result in a loss of file access authority rather than a gain.

Working with default ACLs

To facilitate management of ACLs, you can define a default ACL in a directory; it will then be automatically inherited by an object.

- The file default ACL is copied to a newly created file as its access ACL. It is also copied to a newly created subdirectory as its file default ACL.
- The directory default ACL is copied to a newly created subdirectory as both its access ACL and directory default ACL. You can modify or delete inherited ACLs later.

Default ACLs have the same format as access ACLs.

Examples:

1. Define a directory default ACL for the directory named **/u/ProjectX**.

```
setfacl -m default:group:admins:r-x,default:group:dirgrp:rwx /u/ProjectX
```

The syntax is similar to the earlier examples, but entries contain an extra qualifier to designate the directory default ACL. The groups named **admins** and **dirgrp** will automatically get access to any new subdirectories created within **/u/ProjectX**. Note that the creation of a default ACL will not grant access to directories that already exist.

2. Display the directory default ACL created in Step 1.

```
getfacl -d /u/ProjectX
#file: /u/ProjectX
#owner: TCPAUTO
#group: SYS1
default:group:ADMINS:r-x
default:group:DIRGRP:rwx
```

The **-d** option says to display only the extended ACL entries in the directory default ACL.

3. Define a file default ACL for the directory named **/u/ProjectX**, and all of its subdirectories.

```
setfacl -m fdefault:group:admins:r--, \
        fdefault:group:dirgrp:rw- $(find /u/ProjectX -type d)
```

The extra entry qualifier in this case designates the file default ACL. The groups named **admins** and **dirgrp** will automatically get access to any new files created within the **/u/ProjectX** subtree. Note that the creation of a default ACL will not grant access to files that already exist.

4. Display the contents of all of the ACL types for the directory named **/u/ProjectX**.

```
getfacl -adf /u/ProjectX
#file: /u/ProjectX
#owner: TCPAUTO
#group: SYS1
user::rwx
group::r-x
other::r-x
user:JOE:--x
fdefault:group:ADMINS:r--
fdefault:group:DIRGRP:rwx
default:group:ADMINS:r-x
default:group:DIRGRP:rwx
```

This example requests the access ACL (the **a** option), the directory default ACL (the **d** option), and the file default ACL (the **f** option). The base permission bits are displayed when the **a** option is specified (or defaulted).

Recommendation: Analyze your HFS space utilization before implementing default ACLs in your file system. If you use both file and directory default ACLs in every directory in the file system, a separate physical ACL is created for every new file and directory. Using an access ACL for every directory will probably not cause concerns about space utilization. However, the same cannot be said of files, especially if the inherited ACLs are large.

Tip: ACLs are not inherited across mount points. Suppose that you have a default ACL defined on the directory **/dir1/dir2**. You decide to create another directory, **/dir1/dir2/dir3**, and use it as a mount point on which to mount another file system. However, if you do so, the root directory of the mounted file system will not inherit the default ACL which had been established at **/dir1/dir2**. If you want the default ACLs of **dir2** to apply to **dir3**, you must copy them to **dir3** after **dir3** has been mounted.

Summary of tasks and their associated commands

Table 8 summarizes the tasks that you might want to do and their associated commands.

Table 8. ACL tasks and their associated commands

Task	Shell Command
Add, delete, or update an ACL	setfacl
Display contents of an ACL	getfacl
Update permission bits	setfacl or chmod
Display permission bits	ls or getfacl
Find out whether files have extended ACL entries	ls
Search for files or directories that have various ACL properties	find
Determine if the file system(s) and security product support ACLs	df
Determine if the file system supports ACLs (_PC_ACL) and also determine the maximum number of ACL entries that the file system can support (_PC_ACL_ENTRIES_MAX)	getconf
Restore ACL information or store the information in an archive	pax or tar
Preserve the ACLs for files and directories	cp , mv
Test files and directories for extended ACL information. Also test for directory ACLs and file default ACLs on directories.	filetest , test , [...] and [[...]] reserved-word command

How ACLs are used in file access checks

The algorithm for access checking is up to the security product that is being used. If the physical file system (for example, HFS) supports ACLs, then it uses the **SAFck_access** (IRRSKA00) callable service when passing the ACL to the security product.

If the security product supports ACLs, it applies its own rules to the file access request. RACF uses the permission bits, access ACL, and various UNIXPRIV class profiles to determine whether the user is authorized to access the file with the

requested access level. See the section on protecting file system resources in *z/OS Security Server RACF Security Administrator's Guide* for details on how RACF uses ACLs when enforcing file security.

Auditing changes to ACLs

You can audit the creation, alteration, and deletion of ACLs by using SETROPTS LOGOPTIONS for the FSSEC class. The FSSEC class controls auditing for changes to all file security information, including file owner, permission bits, and auditing options. See *z/OS Security Server RACF Auditor's Guide* for more information.

Using ACLs in a sysplex

Using ACLs should be no different on a sysplex client than on a sysplex server system if all the participating systems are running at V1R3 or higher.

In a sysplex environment, all participating nodes must be on a release level that has ACL support. If any of the participating nodes are at a release level that does not contain ACL support and you have enabled the FSSEC class on an up-level node, then files that are protected by ACLs will not be accessible on down-level nodes (assuming that the compatibility APAR has been applied) except perhaps by a superuser or file owner. The APAR is OW50655 for SAF and OW49334 for RACF.

Using security labels

Traditionally, access to z/OS UNIX resources is based on POSIX permissions. With the SECLABEL class active, authorization checks are performed for security labels in addition to POSIX permissions, to provide additional security. Security labels are used to maintain multiple levels of security within a system. By assigning a security label to a resource, the security administrator can prevent the movement of data from one level of security to another within the z/OS UNIX environment.

When the SECLABEL class is active, security labels can be set on z/OS UNIX resources in the following ways:

- When an HFS file system or zFS aggregate is created, the file system root will be assigned the security label that is specified in the RACF data set profile that covers the data set name. If a security label is not specified or if a data set profile does not exist, then a security label will not be assigned to the file system root.
- zFS file systems support the **chlabel** utility which allows the setting of an initial security label on a file or directory. You can use this utility to set security labels on zFS files and directories after they have been created.
- If a directory has been assigned a security label through one of the above steps, then new files and directories created within that directory will inherit a security label as follows:
 - If the parent directory is assigned a security label of SYSMULTI, the new file or directory is assigned the security label of the user. If the user has no security label, then one will be assigned to the new object.
 - If the parent directory is assigned a security label other than SYSMULTI, the new file or directory is assigned the same security label as the parent directory.
- The rules for security label assignment are more extensive when running in a multilevel-secure environment. For more information, see *z/OS Planning for Multilevel Security*.

Symbolic link restrictions

When using security labels, z/OS UNIX restricts the use of certain character strings within symbolic links to allow for dynamic substitution of a user's security label within a pathname.

The character strings '\$SYSSECA/' and '\$SYSSECR/' have special meaning to z/OS UNIX when they appear as the first nine characters in a symbolic link. For more information on the use of these strings, see the section in *z/OS Planning for Multilevel Security* on assigning a home directory and initial program, depending on security label.

Using multilevel security

Multilevel security is a security policy that allows the classification of data and users on a system of hierarchical security levels combined with a system of non-hierarchical security categories.

In a multilevel-secure z/OS UNIX environment, security labels are used as described in "Security labels for zFS files and directories." To set the SECLABEL on z/OS UNIX files and directories, use the **chlabel** utility.

Restrictions:

- Shared HFS does not provide mixed sysplex support for multilevel security. Running a multilevel-secure environment in a mixed sysplex (with systems below V1R5) will result in unpredictable results.
- To back up and restore files while maintaining security labels, you should use DFDSS. You cannot use the **pax** and **tar** commands.

Security labels for zFS files and directories

The zFS file system is the only physical file system with support for security labels in a multilevel-secure environment. The HFS file system does not provide support for security labels in a multilevel-secure environment. There is setup that can allow for the use of HFS file systems in this environment, but capability is limited to read-only access. For more information about multilevel security, see *z/OS Planning for Multilevel Security*.

Auditing access to files and directories

The security auditor uses reports formatted from RACF system management facilities (SMF) records to check successful and failing accesses to kernel resources. An SMF record can be written at each point where the system makes security decisions.

Six classes are used to control auditing of security events. These classes do not have any profiles. They do not have to be active to control auditing. Use the SETROPTS command to specify the auditing options for the classes. For a list of the classes used for auditing and an explanation of how to specify the audit options, see *z/OS Security Server RACF Auditor's Guide*.

Audit records are always written for the following events:

- When a user not defined as a z/OS UNIX user tries to dub a process
- When a user who is not a superuser tries to mount or unmount a file system

You cannot turn off these audit records.

You can also specify auditing at the file level in the file system. Activate this option by:

1. Specifying **DEFAULT** in the class **LOGOPTIONS** on the **SETROPTS** command
2. Using the **chaudit** command to specify audit options for individual files and directories

If you activate auditing for additional levels of file system access, you may generate excessive amounts of SMF Type 80 records.

You can also specify, in a RACF user profile, that all actions taken by the user be audited. Actions taken by superusers can be audited or not, determined by RACF commands. If you are using RACF profiles in the **UNIXPRIV** class to control certain superuser functions, you can use those same profiles to audit those superuser functions.

Specifying file audit options

Specify file audit options using the ISPF shell, or a **chaudit** command. The command can be used to specify either user audit options or auditor audit options. To specify user audit options, you must be a superuser or the owner of the file. To specify auditor audit options, you must have RACF AUDITOR authority.

If you have AUDITOR authority, you do not need access in the permission bits to:

- Search and read any directory in the file system
- Use the **chaudit** command to change the auditor audit options for any file in the file system

If both user and auditor audit options are set, RACF merges the options and audits all the set options.

For information about audit control, see *z/OS Security Server RACF Auditor's Guide*.

Using sanction lists

Be sure that you are familiar with the activation instructions before using sanction lists. It is possible to unintentionally activate only part of this feature.

You can compile a list to contain the lists of pathnames and program names that are sanctioned by the installation for use by APF-authorized or program controlled calling programs. This file contains properly constructed pathnames and program names as defined in *z/OS UNIX System Services User's Guide*.

Sanction lists contain three separate lists delineated by three keywords:

:authprogram_path

This keyword is the start of a list of directories that is only used in the execution of an hfsload (or C dload), exec, spawn, or attach_exec from an authorized program.

:programcontrol_path

This keyword is the start of a list of directories that is only used in the execution of an hfsload (or C dload), exec, spawn, or attach_exec from an executable that is running program controlled.

afprogram_name

This keyword is the start of a list of program names that are allowed to get control of APF-authorized programs as a result of an exec or spawn. These names are MVS program names.

Formatting rules for sanction lists

Note: You cannot use symbolic links (for example, \$SYSNAME) in sanction lists. They will not work.

You have to follow certain formatting rules when creating sanction lists.

- Only use absolute pathnames.
- Pathnames cannot start with /*.
- Each list element must be on a line by itself, with no comments. Lines are terminated with the newline character, as is consistent with the stepliblist and userdaliastable files. Leading blanks may be on the list element line and are ignored. Use the newline character to delimit a pathname. Trailing blanks are ignored. Other white space is considered part of the pathname.
- Follow standard z/OS UNIX path naming conventions.
- You must follow standard MVS program naming conventions.
- Encode the sanction list file in the IBM-1047 code page.
- You can include comment lines in the list. Each comment line must start with /* and end with */. They cannot be on the same line with any other type of line.
- Do not enclose the pathnames or program names in quotation marks.

The tags **:authprogram_path**, **:programcontrol_path**, and **:apfprogram_name** must be used to delineate between the different types of sanction lists.

- If there are no tags in the file, then all data in the file is ignored and you will get a parsing error. If a tag is missing, then the subsequent processing of hfsload/dlload, exec or spawn will not change, based on the tag that was missing. The effect of different sanction lists is not cumulative. Once a sanction list is parsed and accepted, the contents provide the only active lists of pathnames and program names for hfsloads, execs, and spawns.
- List elements (pathnames or program names) before a tag are ignored.
- Lines after the last valid entry line (such as a pathname or a program name) are ignored.
- If an **:authprogram_path** tag is present, then all lines following it and up to the next tag are considered to be approved path names from which authorized programs can be invoked.
- If a **:programcontrol_path** tag is present, then all lines following it and up to the next tag are considered to be approved path names from which program controlled programs can be invoked.
- If an **:apfprogram_name** tag is present, then all lines following it and up to the next tag are considered to be approved program names that can get control APF-authorized.
- If specified, the tag must start in column 1.
- The tag names are not case-sensitive.
- The list element names (for example, the pathnames and program names) are case-sensitive.

If the file does not follow these formatting rules, the sanction lists may not be recognized properly and various functions relating to the attempted use of the lists may fail.

Steps for creating a sanction list

Before you begin: You need to know what directories and what programs are to be set into this file. You can partially construct this file and add pathnames and program names as you go along. A partially complete file can be activated and when additional entries are known, this file can be updated. A background task will automatically check this file every 15 minutes for updates and then incorporate them.

You also need to be aware that only one sanction list check is done for each program invocation. Although links in directories are supported, sanction list processing only performs one check. This check uses the pathname or program name that was specified by the user.

Tip: The installation can construct the sanction lists with link pathnames or actual pathnames, or both. The decision depends on how the site would like the users to invoke the programs. For example, if the actual directory is in the sanction list instead of the directory that contains the link, and the associated program is invoked via the link, the program would not be executed. The program is only executed if the directory where the link was defined or resides is specified in the sanction list and the associated program is invoked via the link. Alternatively, both the actual directory and directory where the link resides could be placed in the sanction list. This would give users the option of invoking the program either way. Likewise, if only the actual directory was placed in the sanction list, the user would be forced to use actual pathnames and not links.

Perform the following steps to create a sanction list.

1. Create a sanction list, following the rules listed in “Formatting rules for sanction lists” on page 103. You can cut and paste the following sample.

```
/*
/* *****
/* Name: Sample authorized program list
/*
/* Description: Contains lists of approved directories and
/*               program names from which privileged programs
/*               may be invoked
/* *****
/*
/* Authorized program directories
/* *****
:authprogram_path
/bin/test
/bin/test/beta

/* *****
/* Program control directories
/* *****
:programcontrol_path
/in/test/specials

/* *****
/* APF authorized programs
/* *****
:apfprogram_name
PAYOUT
```

2. Name the sanction list.

Recommendation: The pathname of the sanction file should be `/etc/authfile`, in keeping with IBM's strategy to place all customized data in the `/etc` directory.

When you are done, you have created a sanction list. To activate it, see “Steps for activating the sanction list.”

Recommendation: Only users with superuser authority should be given update access to sanction lists.

Steps for activating the sanction list

Before you begin: You must know what the HFS file name is for your sanction list. This file may or may not exist, or it may not be complete, or both. If this file exists, it should be properly constructed as described in “Formatting rules for sanction lists” on page 103 even though it may not be complete.

Perform the following steps to activate the sanction list.

1. Activate the sanction list processing by specifying a value for `AUTHPGMLIST`. If you do not specify a value, the sanction list will not be processed. Base your choice on your particular situation.

If you choose this method . . .	Then . . .
Use the <code>AUTHPGMLIST</code> statement in <code>BPXPRMxx</code> . The sanction list may or may not have already been set up.	Customize the <code>BPXPRMxx</code> parameter. Example: <code>AUTHPGMLIST('/etc/authfile')</code>
Use <code>SETOMVS</code> .	Issue the <code>SETOMVS</code> command. Example: <code>SETOMVS AUTHPGMLIST='/etc/authfile'</code> Tip: To turn off sanction list checking, issue: <code>SETOMVS AUTHPGMLIST=NONE</code>
Recommendation: You should already have set up the sanction list. Otherwise, you will get an error message warning you that the file does not exist. The pathname, however, will be set. If you issue the same command with the same file name, you will not get an error message. The <code>DISPLAY OMVS</code> command will show the <code>AUTHPGMLIST</code> parameter being set. This file name will be used by the background task to check whether a sanction list has been created or updated.	

If you choose this method . . .	Then . . .
A nonexistent sanction list. Recommendation: This feature should only be used in those instances where it is imperative that the sanction list not exist prior to its activation. It is possible to set the sanction list value and forget that the sanction list has not been completely set up. The system may appear to be operating with sanction list processing, but in fact it is not. The background task will routinely check for the nonexistent file, but sanctioning will not be occurring for spawns, execs, and so on. This sanction list file must be set up for sanctioning to occur. The background task will not warn that the sanction list is nonexistent.	Use either method described in this table (customize the BPXPRMxx parmlib member or use SETOMVS).

-
2. If the sanction list has not already been created (see “Steps for creating a sanction list” on page 104), create one now.
-

When you are done, you have activated sanction list processing. A background task will sweep in the background every 15 minutes for updates. Its only job is to check for the sanction list, and if it is there, to process it. Alternatively, if a change needs to be activated sooner, you can use SETOMVS or SET OMVS =(xx), where xx specifies which BPXPRMxx file is to be used to reset the various z/OS UNIX parameters.

Tip: You can turn off sanction list checking with the SETOMVS command:

```
SETOMVS AUTHPGMLIST=NONE
```

Notes:

1. If the sanction list has not been created when the system is IPLed, you can create it later and then use the SETOMVS command to dynamically add it. However, you should be careful because you will not get a message saying that the sanction list file does not exist, although z/OS UNIX will continue to check every 15 minutes.
2. If the sanction list was created before the system is IPLed, and there are errors, the sanction list processing is disabled.
3. If the AUTHPGMLIST statement in the BPXPRMxx parmlib member contains a nonexistent value, you will not get an error message.
4. If the sanction list is running on the system, you will get error messages when you try to run program-controlled or APF-authorized programs that are not in the sanction list. You will have to add them to the sanction list.

Maintaining the security level of the system

After you set up a secure environment for your system, you must ensure that it stays secure.

Steps for maintaining the security level of the system

Before you begin: You need to have set up a secure environment for your system.

Perform the following steps to ensure that the system stays secure.

1. Check each program that you want to introduce into the system. Add a program only if you are certain that it will not lower the level of security.

2. For users of the system, set up rules for:
 - Sharing data in files
 - Specifying permission bits when creating files or using the **chmod** command or **chmod()** function

3. Require that users set the permission bits for their files to deny access to all users except themselves, as the file owners.

4. Protect all local data sets with a RACF profile that specifies UACC(NONE). Only administrators with responsibility for creating, restoring, or dumping local data sets should be permitted to this profile.

When you are done, you have taken steps to ensure that your system stays secure.

Defining the OMVSAPPL profile for the APPL class

If the APPL class for the security product is active, the OMVSAPPL profile can be defined to allow only certain users to sign on to that application. For example, if you do not want all of your users to use z/OS UNIX, you can activate the APPL class and create a profile for OMVSAPPL with an access list that contains only those users who are allowed to use z/OS UNIX.

z/OS UNIX specifies OMVSAPPL for the APPLID parm on its client ACEE calls, which are passed to RACROUTE REQUEST=VERIFY via the APPL= parameter. With the APPL value, you can tell what application the user is entering the system through, and can restrict that access, if desired.

The following services specify OMVSAPPL as the profile name for the APPL class when a new security environment is created. You can restrict access to OMVS via the APPL class only for these services:

- pthread_security_np
- _security
- osenv
- _passwd when there is no password change specified
- _paswd when the calling process never called pthread_security_np

In general, you should not need to define a profile for OMVSAPPL unless you have a generic profile (*) that prevents access to applications that do not have a more specific profile defined.

Setting up TCP/IP security

Rules:

1. The TCP/IP started task's user ID and its default group must both have an OMVS segment defined. The user ID, assigned using ICHRIN03 or the STARTED class, must have UID(0). For information on defining a z/OS UNIX user to RACF, see "Defining z/OS UNIX users to RACF" on page 60.
2. Other TCP/IP tasks such as **ftp** and **routed** must be assigned a RACF user ID using ICHRIN03 or the STARTED class. If **ftpd** and **routed** use a different started task user ID from the TCP/IP user ID, they must have UID(0) and HOME('/').

Selecting a security level for the system

If you run daemons and servers, you need to set up the appropriate security for them. There are two levels of privileges available for servers and daemons: UNIX level and z/OS UNIX level. In providing security, you need to understand the differences between the two levels. For discussions of the levels, see:

- "Establishing the correct level of security for daemons" on page 291
- "Establishing the correct level of security for servers" on page 322

Chapter 6. Managing the hierarchical file system

Overview

Either the data administrator or the system programmer must manage the hierarchically organized data that the system and its users will use. This overall structure of data is called the *hierarchical file system (HFS)*. It consists of the root file system and all the file systems that are added to it.

To gain some background for performing HFS-related activities, read “Hierarchical file system concepts.”

During the first quarter of 2000, PTFs removed the requirement that HFS data sets reside on SMS-managed volumes. HFS data sets still need to be cataloged in the master or user catalog in order for the HFS data sets to be mounted by z/OS UNIX. However, you do not need to catalog the HFS data sets if you plan on dumping them using DFSMSdss™. The PTFs you need to apply are among those that are listed in the driving system requirements in *z/OS and z/OS.e Planning for Installation*.

If you are using the shared HFS capability in a sysplex, you know that the term “root HFS” is called the version HFS. Throughout this chapter, you should think of the version HFS when you see “root HFS”.

If you want to take advantage of the automount facility, see Chapter 8, “Using the automount facility,” on page 147.

In this chapter

This chapter covers the following subtasks.

Subtask	Associated procedure (see . . .)
Mounting file systems	“Steps for mounting file systems” on page 116
Customizing the mail utility	“Steps for customizing the mail utility” on page 125
Recovering from file system problems with the root	“Steps for recovering from file system problems with the root” on page 138

Hierarchical file system concepts

The HFS consists of the following:

- HFS 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 that contain files, other directories, or both. Directories are arranged hierarchically, in a structure that resembles an upside-down tree, with the root directory at the top and the branches at the bottom. The **root** is the first directory for the file system at the top of the tree and is designated by a slash (/).
- Additional local or remote file systems, which are mounted on directories of the root file system or of additional file systems.

z/OS UNIX files are organized in a hierarchical file system (HFS), 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 following illustration 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; in this case, z/OS BookServer and z/OS TCP/IP.

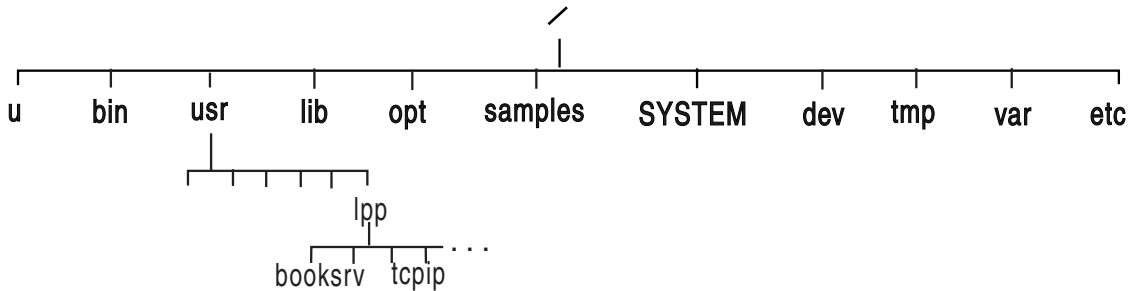


Figure 9. Logical view of the HFS for the end user

Except for the direction of the slashes, the hierarchical file system is similar to a Disk Operating System (DOS) or an OS/2® file system. To MVS, the z/OS UNIX files form a hierarchy that is a collection of data sets of the type “HFS”. Each HFS data set resides on direct address storage. DFSMS manages the HFS data sets and the physical files.

As of OS/390 Release 7 and DFSMS 1.5, HFS data sets can span volumes.

HFS data sets have multiple volume support. This allows an HFS data set to span a total of 59 volumes with limits of 123 extents per volume and 255 extents across all volumes. The maximum size of an HFS data set is the following:

- 2^{31} pages (4K bytes/page) = 2^{43} bytes

The volume and extent limits are MVS limits, not HFS limits.

Required file system structure

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. IBM recommends 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**, which 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. As described in “Establishing an /etc file system for a new release” on page 12, after you complete instructions for a ServerPac or CBPDO installation, you will have an **/etc** file system in its own HFS data set.
- **/dev**, which 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**, which contains temporary data that are used by products and applications. **/tmp** is created empty, and temporary files are created dynamically by different elements and products. You have the option of mounting a temporary file system (TFS) on **/tmp**. For more information, see Chapter 16.
- **/var**, which contains dynamic data that is used internally by products and by elements and features of z/OS. Any files or directories that are needed are created when code is executed or customized. An example of this is caching data. In addition, you can be assured that IBM products will only create files under **/var** when code is executed or customized.

z/OS UNIX System Services User's Guide discusses the concepts of file systems, directories, and files. You should understand them before reading further in this document.

Command differences due to symbolic links

Starting in OS/390 Release 9, certain directories like **/etc**, **/dev**, **/tmp**, and **/var** have been converted to symbolic links. Some shell commands have minor behavioral differences when referring to symbolic links than for regular files or directories. For example, **ls** does not follow symbolic links by default. Prior to Release 9, **/etc** was a directory, so **ls /etc** would display all files in **/etc**. Beginning in Release 9, **/etc** became a symbolic link, so **ls /etc** displays only the symbolic link. In this case, it is **/etc**.

In order to follow symbolic links, you must specify **ls -L** or provide a trailing slash. For example, **ls -L /etc** and **ls /etc/** both display the files in the directory that the **/etc** symbolic link points to.

Other shell commands that have differences due to symbolic links are **chmod**, **du**, **find**, **pax**, **rm**, and **tar**.

While these behavioral changes should be minor, users can tailor command defaults by creating aliases for the shell command. For example, if you want **ls** to follow symbolic links, you could issue the command:

```
alias ls="ls -L"
```

Aliases are typically defined in the users' ENV file. Refer to *z/OS UNIX System Services Command Reference* for details on the **alias** command.

Note: After you establish the alias, **ls** will follow all symbolic links.

An administrator can put **alias** commands in **/etc/profile**, which could affect all users' login shells. IBM does not recommend this, because changing the default behavior in **/etc/profile** might produce unexpected results in shell scripts or by shell users.

Recommended file system structures for user directories and files

For users, you should logically mount other HFS data sets on the root file system. You should also have your users place their directories and files in the mounted file systems. Separate user file systems offer several advantages:

- They improve storage management because the system administrator only needs to allocate data sets that are large enough to accommodate the needs of individual users.
- They enable failure isolation because the system administrator can unmount the user file system that caused an error without affecting other users' data or causing z/OS UNIX to fail.
- They relieve the contention for system resources that could occur by having multiple users in a single file system.

Name each user's home directory **/u/userid** where **userid** is the user ID in lowercase.

Keep system HFS data sets separate from user HFS data sets by segregating the HFS data sets on different volumes.

Recommendation: Use the automount facility, which is described in Chapter 8, "Using the automount facility," on page 147.

Using the Network File System (NFS)

A workstation user connected to a host through TCP/IP can mount all or part of an HFS that is at the host so that it appears as part of the user's local file system. A combination of the TCP/IP server and Network File System (NFS) makes this possible.

If you are using the NFS server, you can make both traditional MVS data sets and hierarchical files appear as part of the user's workstation file system. The user can create, delete, read, write, and otherwise treat the host-located files as an extension of the workstation's own file system. ASCII-EBCDIC conversion for single-byte text files is performed automatically by means of default standard conversion tables. (NFS does not provide conversion of double-byte text files.)

Using the NFS client, you can access hierarchical files and MVS data sets on other z/OS systems. You can also access hierarchical files on any system with an NFS server and the proper protocol support.

For more information about NFS, refer to:

- *z/OS Network File System Customization and Operation*
- *z/OS Network File System User's Guide*

For more information about TCP/IP, refer to:

- *z/OS Communications Server: IP Migration and Exploitation*
- *z/OS Communications Server: IP Configuration Reference*
- *z/OS Communications Server: IP User's Guide and Commands*

Creating a hierarchical file system

If you are a systems programmer, you will create and manage the hierarchical file system. This involves some of the following tasks:

- Allocating the root file system or the root HFS data set.
- Mounting the root file system by placing a ROOT statement in the BPXPRMxx parmlib member. During initialization, the system mounts the file systems in the ROOT statement and in all MOUNT statements in BPXPRMxx.

You can also change the active MOUNT attributes of the root without having to re-IPL by using the TSO/E MOUNT and UNMOUNT commands. However, if you have any users logged on or applications running, this method can be disruptive.

- Adding directories to the root file system. You can use an empty directory as a mount point for a file system that you are mounting.
- Adding MOUNT statements in BPXPRMxx for all file systems that you mount so that they will be mounted whenever the system is IPLed.

If a file system is not mounted, the user will not have access to it. The BPXPRMxx parmlib member can contain MOUNT statements for each of the file systems that you created. You can also create a REXX exec that contains multiple MOUNT statements, one for each of the file systems.

Various methods for mounting are:

- Using the automount facility
- Using a TSO/E CLIST or REXX exec
- Issuing the TSO/E MOUNT command from **/etc/rc** using the **tso** shell command. For example:

```
/bin/tso "mount file(OMVS.HFS.D96)
mount('/u/d96') type(hfs) mode(read)"
```

- Running the REXX exec **/usr/sbin/mount** from **/etc/rc**. For example:

```
/usr/sbin/mount -f OMVS.HFS.D96 /u/d96
```

- Using the **/sample/samples/mountx** utility
- Using the **mount** shell command
- Using an automation product such as NetView for mounting a file system

Using uppercase and lowercase letters in filenames and pathnames

In the JCL used for files, the filenames and pathnames can be in lowercase letters, in uppercase and lowercase letters, or in uppercase letters. The case of letters is important. *The editor used to create the JCL must not change the filenames and pathnames into uppercase.*

Allocating an HFS data set for the root file system

In open systems analogous to z/OS UNIX, allocation may be called *making the file system*. The root file system must be allocated by a user who has an UID of 0, indicating superuser authority. To create HFS data sets, a security product that supports the SAF calls made during the system processing must be running.

Tip: All newly allocated file systems have permission bits set at 700.

Figure 10 on page 114 shows a sample job that defines the root HFS data set in an MKFS DD statement. The allocation specified in this sample does not reflect the amount of space needed for the root file system. For exact size information, consult *z/OS Program Directory*. When you specify space for the HFS data set, you must provide a nonzero value for the directory space parameter, which is not used. Or you can specify DSORG=PO to create a data set with partitioned organization.

Later, during customization, put the DSNAME of the root file system in the ROOT statement in the BPXPRMxx parmlib member. For an example, see Figure 5 on page 25.

```

//OMVSXX JOB
//STEP03 EXEC PGM=IEFBR14
//MKFS DD DSN=OMVS.ROOT,
//        SPACE=(CYL,(40,1,1)),DCB=(DSORG=PO),
//        DSNTYPE=HFS,
//        DISP=(NEW,CATLG,DELETE),
//        STORCLAS=STANDARD

```

Figure 10. Job to allocate the HFS data set

Other MVS data sets can reside in available parts of the volume containing the HFS data set. You can also use the ISPF shell, or the TSO/E ALLOCATE command to create an HFS data set.

See the following:

- For JOB, EXEC, and DD statements, see *z/OS MVS JCL Reference*.
- For BPXPRMxx guidance information, see “Customizing the BPXPRMxx parmlib members” on page 24.
- About setting up the root file system structure, see “Required file system structure” on page 110.
- For BPXPRMxx reference information, see *z/OS MVS Initialization and Tuning Reference*.

HFS data sets can be allocated by systems other than the one on which the data set will be used, as long as the allocating system has the correct level of DFSMS. The system where the HFS data set will be used must share the catalog with the allocating system or must have a catalog entry for the same HFS data set name.

Defining the root file system

Use the ROOT statement in the BPXPRMxx member of SYS1.PARMLIB to define which HFS data set is the root file system. Figure 5 on page 25 shows the sample BPXPRMxx parmlib member. The ROOT statement is as follows:

```

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

```

The root file system is the starting point for the overall HFS file structure. It contains the root directory and any related HFS files or subdirectories.

What happens when file systems are mounted?

After you mount a new file system for the first time (as described in “Mounting file systems” on page 116), you need to change the owner and group owner. This section explains why this step is necessary.

To begin with, the hierarchical file system is used to store data and organize it in a hierarchical way by using file system entries such as directories and files. These file system entries have certain attributes, such as ownership, permission bits, and access 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.

Mounting a file system creates a *binding* for the duration of the mount. The binding is between a directory that is already in the file system hierarchy, called the *mount*

point, and the entry point into the file system about to be mounted, called the *root* of this file system. The mount point directory and the root are connected until unmount time.

See Figure 11 to see what happens when Jane's file system is mounted on directory `/u/jane`.

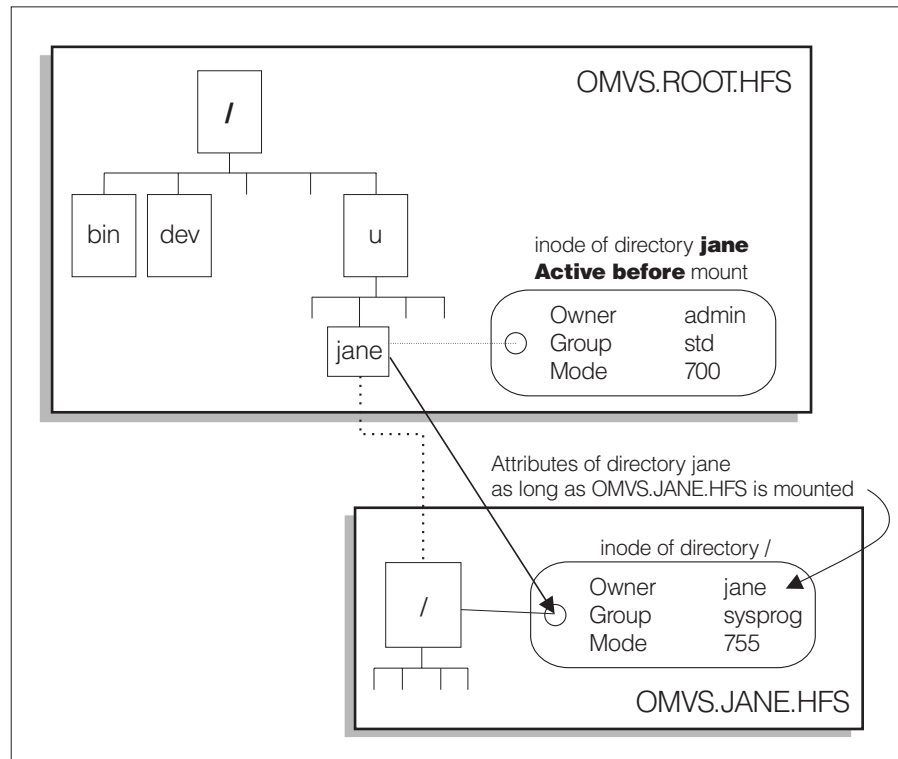


Figure 11. Mounting a file system

The root directory of a file system, like any other entry, also has attributes, but the directory does not have a name. At mount time, the mount point directory lends its name to the root directory of the file system that is to be mounted. The root, however, keeps its attributes. Logically, the directory (which is an entry in another directory, one level up the hierarchical tree) no longer points to its own inode. Instead, it points to the inode of the mounted root. Thus, the attributes and the content of a directory are hidden as long as a file system is mounted on it.

A newly created file system is empty except for an unnamed root directory. Because this directory needs attributes, which will be stored in its inode, the creating routine must set some start values. For HFS, the DFSMS allocation routines set ownership to the user who allocates the data set. The routines also set the file mode to 700 in order to ensure that only this user can access it until that user permits other access.

When you create file systems for other users, you must change the attributes of the root directory of the new file system. To do so, mount the new file system, and then change the attributes. You need to change the attributes only once because your changes are saved in the root's inode. Assuming that the file system was mounted on a directory called *mount-point-name*, do the following:

1. For a user file system, change the ownership and leave it up to the user to set the mode:

```
chown target-userid:target-groupid mount-point-name
```

2. For public file systems, change the file mode and assign the correct ownership:

```
chmod 755 mount-point-name
```

```
chown system-userid:system-groupid mount-point-name
```

The *system-userid* and *system-groupid* must match the purpose of the file system. The same applies to the mode. A mode of 755 allows anyone to make this directory the current working directory, but only the owner can write to it. For other situations, a mode of 750 may be more appropriate.

Remember that the root of a file system always keeps the attributes that it had at **unmount time**. The same attributes are used again when the file system is mounted later. The attributes do not depend on the actual mount point.

Mounting file systems

After installation is complete, you need to create HFS data set file systems and mount them to your root file system or somewhere else in the hierarchy.

Steps for mounting file systems

Before you begin: You need to know that the mount point should be an empty directory. If it is not, then its contents will be hidden for the duration of any subsequent mounts.

Perform the following steps to mount a file system.

1. Build a directory in the root file system. A directory can be used as a mount point for a file system.

To build the directory, use one of the following:

- The TSO/E MKDIR command interactively; in an in-stream data set in the JCL, such as SYSIN; or in a CLIST or REXX exec.
- The **mkdir** shell command.
- The TSO/E ISHELL command.
- The MKDIR keyword in a ROOT or MOUNT statement in the BPXPRMxx parmlib member.

-
2. Allocate an additional HFS data set, using one of the following:

- The TSO/E ALLOCATE command
- The TSO/E ISHELL command
- The JCL DD statement
- ISPF Option 3.2

Example: Sample TSO/E ALLOCATE commands are:

```
ALLOCATE DATASET('OMVS.USER.JOE') DSNTYPE(HFS) SPACE(5,5) DIR(1) CYL
ALLOCATE DATASET('OMVS.USER.JANE') DSNTYPE(HFS) SPACE(5,5) DIR(1) CYL
```

Tip: To allocate an HFS data set in a single extent, use the following ALLOCATE command:

```
ALLOCATE DATASET('OMVS.USER.SAM') DSNTYPE(HFS) SPACE(5,0) DIR(1) CYL
```

-
3. Free the data sets.

Example: The FREE commands for the HFS data sets are:

```
FREE DATASET('OMVS.USER.JOE')
FREE DATASET('OMVS.USER.JANE')
```

-
4. Logically mount the new file system in the directory of an existing file system by using the TSO/E MOUNT command under a user with mount authority.

Example: The directory **/u/joe** is a mount point for OMVS.USER.JOE and **/u/jane** is a mount point for OMVS.USER.JANE. The MOUNT commands for the mounted file systems are:

```
MOUNT FILESYSTEM('OMVS.USER.JOE') TYPE(HFS) MOUNTPPOINT('/u/joe')
MOUNT FILESYSTEM('OMVS.USER.JANE') TYPE(HFS) MOUNTPPOINT('/u/jane')
```

When you are done, you have mounted a file system.

Rule: After you mount the new file system for the first time, change the owner and group owner. These values are saved in the new file system and are reused when the file system is remounted later. Use the **chown** command to set these values. (You may need superuser authority to issue the **chown** command, depending on your installation.)

Example: For the **/u/joe** directory, to set the user name and group name (if they have already been defined to the security product), issue:

```
chown joe:devgrp /u/joe
```

Tip: You can also use the ISPF shell to set the permissions.

Restrictions on mounting file systems

The restrictions on mounting file systems are:

- 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.
- These restrictions apply to file systems that are mounted by systems in a sysplex:
 - Systems earlier than OS/390 R9 and systems not participating in shared HFS capability can mount file systems that will be shared in read-only mode.
 - Systems at OS/390 R9 or later, participating in shared HFS capability, can mount file systems that will be shared in read/write mode.
 - For systems earlier than OS/390 R9 to share file systems with systems using shared HFS capability, the file systems must be mounted in read-only mode on all systems.
- The HFS data sets in the same file hierarchy cannot have the same name even if they are mounted on different mount points. This remains true whether real names or alias names are specified on the FILESYSTEM operands in BPXPRMxx or on the MOUNT command. If two HFS data sets have the same name, they cannot be mounted.
- There is an upper limit to the number of HFS data sets that can be mounted at one time in your system. For planning purposes, about 1K of storage is consumed below the 16M line for each mounted HFS file system. You can limit the amount of storage that is consumed. To limit the amount, use the timeout capabilities of automount so that file systems are unmounted when they are not

being used. This storage below the line is used for the data set allocation. If storage is not available and another data set allocation is requested, the system may be placed in a nonrestartable wait state.

Managing file systems

DFSMS manages the location of all HFS data sets (file systems) on volumes. However, a file system can outgrow the space on its volume and need more space. Or activity in a file system can become so great that it slows response time. In these cases, the file system needs to be managed.

As of OS/390 Release 7, HFS data sets can span volumes. As users add files and extend existing files, each data set can increase in size to a maximum of 123 extents if secondary extents are specified in the allocation. The system programmer can:

- Remove other data sets from the volume on which the full volume condition resides.
- Move individual HFS files and subtrees to other volumes.
- Move the entire full file system to another HFS data set.

The storage administrator or system programmer can monitor the space in a file system by mounting an HFS with the FSFULL parameter. For example, the following would cause HFS to issue message IGW023A when the file system is 70 percent full. Then it would issue another message when the file system is 80 and 90 percent full:

```
mount parm('FSFULL(70,10)')
```

Reducing the size of the file system

If the file system becomes too big for the volume, you can try to reduce the size of the file system:

- Create a new file system on another volume and move some files from the full file system to the new file system. Mount the new file system onto the previously full file system.
- Move a subtree from the active file system into a new file system on a different volume. Mount the new file system onto the now-empty directory that was the head of the subtree. Accesses are divided between two volumes.

Moving a subtree, rather than individual files, retains the hierarchical structure of the file system.

Increasing the size of the file system

Another approach to making more space available for the file system is to move the entire full file system to another HFS data set, as follows:

1. Have an authorized user enter a TSO/E UNMOUNT command to logically unmount the file system.

Tip: The REXX exec `/usr/sbin/unmount` performs essentially the same functions that the UNMOUNT statement performs. You can run it from the shell.

2. Use the DFSMSdss dump utility to logically dump the old file system to a sequential data set.
3. Rename the old file system.
4. Preallocate a new HFS data set with a larger size and give it the original file system name.

5. Use the DFSMSdss restore utility to restore the old file system to the new data set.
6. Have an authorized user enter a TSO/E MOUNT command to logically mount the new file system. You can also run the REXX exec **/usr/sbin/mount** from the shell.
7. After you have checked the new file system, you can delete the old file system and the corresponding sequential data set.

You can also use the **confighfs** command to manage or expand the HFS. It resides in **/usr/lpp/dfsms/bin/confighfs**.

Removing unnecessary files from directories

You can use the **skulker** z/OS UNIX shell script to remove files that are older than a specified number of days from any directory. It can be run manually or invoked automatically using **cron**.

The **skulker** shell script, which is located in **/samples**, should be copied. You can modify it to suit your particular needs. Possible locations for the script include **/bin** or **/usr/sbin**, especially if **skulker** is to be run from an UID(0) program. If **skulker** is to be run by users, a locally created directory called, **/usr/bin** is a possibility, but ensure that the sticky bit is on in that directory.

For more information about **skulker**, see *z/OS UNIX System Services Command Reference*.

Improving accesses to file systems

If activity for a file system becomes so extensive that accesses are slow, do one of the following:

- Move the file system to a volume chosen for speed because it has, for instance, a faster channel or buffered controller.
- Move a subtree from the active file system into a new file system on a different volume. Mount the new file system onto the now-empty directory that was the head of the subtree. Accesses are divided between two volumes.

Moving a subtree, rather than individual files, retains the structure of the file system.

Unmounting file systems

To unmount all active file systems, issue the following operator command:

```
F BPXOINIT,SHUTDOWN=FILESYS
```

It unmounts the file systems on the system that the command was issued from.

Read-only BPAM access

You can set up read-only basic partitioned access method (BPAM) access to UNIX files, including HFS, zFS, NFS, and TFS files. Each z/OS UNIX directory is treated as if it were a PDSE or PDS directory. For more information about BPAM access to z/OS UNIX directories, see *z/OS DFSMS: Using Data Sets*.

Mounting your root HFS for execution

This topic describes the mounting of the root HFS data set. (In a shared HFS environment, the “root HFS” is known as the version HFS.) The root HFS data set contains system code and binaries, including the **/bin**, **/usr**, **/lib**, **/opt**, and **/samples** directories. These directories contain files that are installed and serviced by SMP/E.

Rules:

- The **/etc**, **/dev**, **/tmp**, and **/var** directories must have their own HFS data set, separate from the root HFS data set, as of OS/390 R9.
- The root HFS must be mounted in read/write mount mode because when you install and service the files in the root HFS, you will be adding and deleting files and directories from the HFS. After you install and service the root HFS, you can change its mount mode to read-only. Or, you can leave the root HFS in read/write mode.

Deciding how to mount your root HFS for execution

This section helps you decide whether to keep your root HFS read/write or change it to read-only for execution. Table 9 describes the benefits and drawbacks of the two mount modes for the production system's root HFS.

Table 9. Comparing read-only and read/write mode for the execution system's root HFS

Mount Mode	Benefits	Drawbacks
Read-write	• You can create directories or files dynamically in the root HFS. • You do not have to perform the actions listed in Table 10 on page 121.	• Poorer performance for SYSPLEX HFS operations because HFS I/O must be directed between system images in a sysplex. • Someone might modify files or directories inadvertently.
Read-only	• Better performance for sysplex HFS operation because HFS I/O will not need to be directed between system images in a sysplex. • No one can modify directories or files within the root HFS inadvertently.	• No one can create new directories or files dynamically in the root HFS. • You must perform the actions listed in Table 10 on page 121. • You have extra tasks related to leaving some directories in read-write mode such as /dev, /tmp, /etc, and /var, when these do not have their own HFS data set separate from the root HFS data set).

To decide whether you should leave the root HFS read/write or change it to read-only, use the information in Table 9, and any other information that you may have. IBM recommends that you mount the root HFS in read-only mode. If you are mounting the root HFS in a shared HFS environment, the recommendation is a very strong one.

Leaving the root HFS mounted in read/write mode

To leave the root HFS mounted in read/write mode, make sure that the **MODE** parameter of the **BPXPRMxx** parmlib member has been specified **RDWR**:

```

ROOT FILESYSTEM('omvs.hfs.ROOT')    /* z/OS Root Filesystem    */
      TYPE(HFS)                      /* filesystem type HFS      */
      MODE(RDWR)                     /* mounted for readwrite    */

```

Post-installation actions for mounting the root HFS in read-only mode

First, run the FOMISCHO sample job from SYS1.SAMPLIB if you use the **mesg**, **talk**, and **write** utilities and the **uucp** function. This may require updating **/etc/rc**.

If you want to mount the root HFS read-only, look at Table 10 to see if you have to take any actions. The table includes all z/OS base and optional elements that install into the HFS.

These actions can be taken in any order, and do not need to be performed in a certain sequence.

Table 10. Required post-installation activities for mounting a read-only root HFS

Element or Function	Required Action
Library Server	See the section on advanced customization parameters in <i>z/OS Program Directory</i> .
C/C++ Open Class [®] Library	No required actions.
Communications Server IP	No required actions.
Cryptographic Services - Open Cryptographic Services Facility (OCSF)	No required actions.
DCE Application Support	No required actions.
DCE Base	No required actions.
DFSMS	No required actions.
Distributed File Service	No required actions.
Encina [®] Toolkit Executive	No required actions.
Hardware Configuration Definition	No required actions.
IBM HTTP Server	For the directives that must be changed, see <i>HTTP Server: Planning, Installing, and Using</i> , which can be found from the web at: http://www.ibm.com/software/webservers/appserv/library/index.html
Infoprint [®] Server	Change ownership of the Infoprint Server files to the Infoprint Server GID by running the aopsetup customization script. Also, a separate file system must be mounted read/write on the /var mount point. See <i>z/OS Infoprint Server Customization</i> for more information.
Language Environment	No required actions.
Network File System (NFS)	No required actions.
z/OS UNIX System Services Application Services	If you are using shared HFS, move the files associated with the cron , uucp and mail utilities from the root HFS. For more details, see “Customizing cron, uucp, and mail utilities for a read-only root HFS” on page 122.
SecureWay [®] Security Server - DCE	No required actions.
SecureWay Security Server - Firewall Technologies	No required actions.
SecureWay Security Server - LDAP Server	No required actions.
SecureWay Security Server Enhanced Plug-In (OCEP)	No required actions.
System Secure Sockets Layer (SSL)	No required actions.

Table 10. Required post-installation activities for mounting a read-only root HFS (continued)

Element or Function	Required Action
Text Search	Run the <code>/usr/lpp/TextTools/install/imocust</code> procedure. For the procedure, see <i>z/OS Text Search: Installation and Administration for the Text Search Engine</i> .
Tivoli® Framework Endpoint	Copy the <code>lcf.sh</code> script file from <code>/usr/lpp/Tivoli/lcf/bin/os390/mrt/</code> to the <code>/etc/Tivoli/lcf/dat/</code> file.
WebSphere® Application Server	For the procedure, see <i>WebSphere Application Server for z/OS and OS/390</i> , which is available only from the web at: http://www.ibm.com/software/webservers/appserv/library.html
z/OS UNIX System Services Application Services Connection Manager	No required actions.
z/OS UNIX System Services Application Services Process Manager	No required actions.
z/OS UNIX System Services - Integrated Call Level Interface (ICLI)	No required actions.

Mounting the root HFS in read-only mode

After you perform the actions in Table 10 on page 121, update the BPXPRMxx parmlib as follows:

```

ROOT FILESYSTEM('OMVS.ROOT')      /* z/OS Root Filesystem      */
      TYPE(HFS)                    /* filesystem type HFS        */
      MODE(READ)                   /* mounted for read           */

```

With the root HFS mounted in read-only mode, you must define other BPXPRMxx parameters for the directories that remain read/write.

Example: You might also have the following BPXPRMxx entry for `/etc`:

```

MOUNT FILESYSTEM('OMVS.ETC')      /* z/OS etc file system      */
      MOUNTPOINT('/etc')          /* mount at the etc file system */
      TYPE(HFS)                   /* filesystem type HFS        */
      MODE(RDWR)                  /* mounted for readwrite      */

```

Customizing cron, uucp, and mail utilities for a read-only root HFS

Recommendation: With shared HFS support, you should mount the root HFS data set (also known as the version HFS data set) read-only. For more information about the version HFS data set, see Chapter 9.

Before you can mount the root HFS read-only, directories and files that were written to by the **cron**, **uucp**, and **mail** utilities must be moved or redirected out of the root HFS data set and into a directory structure that resides in a read-write mounted HFS data set.

The following instructions are supplemental steps needed to set up these utilities. See *z/OS UNIX System Services Command Reference* for additional customization instructions for configuring these utilities.

Notes:

1. The instructions work for most customer environments. Customers with different customized environments should use this information as a basis to make their respective changes.

2. The instructions are also valid for OS/390 R9 systems that run in a non-shared HFS environment.
3. UID(0) is required to implement these instructions.
4. The instructions are based on the assumptions that the target OS/390 R9 or later system has been IPLed in your test environment, the root HFS data set is mounted on / (slash = the root) in read/write mode, and the ETC HFS data set is mounted on the **/etc** mountpoint in read/write mode.
5. It is also assumed that HFS data sets such as the Sysplex root and the System-specific HFS data sets have not been implemented (that is, they are not being used or mounted) when performing these instructions.

Customizing the cron and uucp utilities

There are three steps for customizing the **uucp** and **cron** utilities. Follow these steps after you have logged in to the OMVS shell environment through TSO, telnet or rlogin.

First Step: Move the contents of the **/usr/spool** directory to under the **/etc/spool** directory. Ensure that you do not have any HFS data sets mounted on **/usr/spool** or any mount point under this directory. If you do, unmount them now. (You can also use **/var/spool** instead of **/etc/spool**.)

1. Create a directory called **/etc/spool**. Issue:
`mkdir /etc/spool`
2. Because spool directories tend to be used heavily, it is good practice to create a new HFS data set and mount it on **/etc/spool**. This step is optional.
3. Change its permission setting to 755. Issue:
`chmod 755 /etc/spool`
4. Change the current working directory to **/usr/spool**. Issue:
`cd /usr/spool`
5. Issue the following **pax** command to copy the contents of the **/usr/spool** directory into **/etc/spool**:
`pax -rw -pe ./ /etc/spool`
6. Change the current working directory to **/usr**. Issue:
`cd /usr`
7. Remove the files and directories under and including the **/usr/spool** directory. Issue:
`rm -fr /usr/spool`
8. Create a symbolic link for **/usr/spool** that points to **/etc/spool**. Issue:
`ln -s /etc/spool /usr/spool`

Second Step: Move the contents of the **/usr/lib/cron** directory to under the **/etc/cron** directory. Ensure that you do not have any HFS data sets mounted on **/usr/lib/cron** or any mount point under this directory. If you do, unmount them now.

1. Create a directory called **/etc/cron**. Issue:
`mkdir /etc/cron`
2. Change its permission setting to 755.
`chmod 755 /etc/cron`
3. Change the current working directory to **/usr/lib/cron**.
`cd /usr/lib/cron`
4. Copy the contents of the **/usr/lib/cron** directory into **/etc/lib/cron**.
`pax -rw -pe ./ /etc/cron`

5. Change the current working directory to **/usr/lib**.
`cd /usr/lib`
6. Remove the files and directories under and including the **/usr/lib/cron** directory.
`rm -fr /usr/lib/cron`
7. Create a symbolic link for **/usr/lib/cron** that points to **/etc/cron**. Issue:
`ln -s /etc/cron /usr/lib/cron`

Third Step: Move and redirect specific **uucp** files that are in **/usr/lib/uucp** to a new directory called **/etc/uucp**. Because executable code are also in this directory, you cannot redirect the entire directory. Instead, you must move specific files.

1. Create a directory called **/etc/uucp**.
`mkdir /etc/uucp`
2. Change its permission setting to 774.
`chmod 774 /etc/uucp`
3. Change the current working directory to **/usr/lib/uucp**.
`cd /usr/lib/uucp`
4. Check to see if the following files exist:
`/usr/lib/uucp/Systems`
`/usr/lib/uucp/Devices`
`/usr/lib/uucp/Dialers`
`/usr/lib/uucp/Dialcodes`
`/usr/lib/uucp/Permissions`
`/usr/lib/uucp/config`

If so, issue:

`ls -al`

5. If none of these files show up in the directory listing, then go to Step 7.
6. If any of the files listed above exist, move them to the **/etc/uucp** directory, by using the **mv** command.

For example, if the file named **SYSTEMS** exists, then you must move it.

`mv -f Systems /etc/uucp/Systems`

Issue the **mv** command for any of the files listed above:

`mv -f xxxxxxxx /etc/uucp/xxxxxxx`

where xxxxxxxx is file name.

7. Create a symbolic link for these files into the **/etc/uucp** directory. Issue the following commands:
`ln -s /etc/uucp/Systems /usr/lib/uucp/Systems`
`ln -s /etc/uucp/Devices /usr/lib/uucp/Devices`
`ln -s /etc/uucp/Dialers /usr/lib/uucp/Dialers`
`ln -s /etc/uucp/Dialcodes /usr/lib/uucp/Dialcodes`
`ln -s /etc/uucp/Permissions /usr/lib/uucp/Permissions`
`ln -s /etc/uucp/config /usr/lib/uucp/config`

At this point you have completed the customization steps for the **uucp** and **cron** utilities. If you unmounted any HFS data sets that were mounted on or below **/usr/lib/cron** or **/usr/spool**, you can mount them now using the same mount point as before.

Customizing the mail utility

This section describes how to customize the **mail** utility.

Steps for customizing the mail utility

Before you begin: You need to first login to the OMVS shell environment through TSO, telnet, or login.

Perform the following steps to customize the **mail** utility.

1. Create a directory called **/etc/mail**.

```
mkdir /etc/mail
```

2. Change its permission setting to 775.

```
chmod 775 /etc/mail
```

3. Change the current working directory to **/usr/mail**.

```
cd /usr/mail
```

4. Check to see if the directory is empty.

```
ls -al
```

If it is empty, go directly to Step 6. You can tell that it is empty if the only files listed are . (dot) and .. (dot dot)

5. If there are files under the **/usr/mail** directory (the output of the **ls -al** command shows more than . and ..), issue the following **pax** command to copy the contents of the **/usr/mail** directory into **/etc/mail**:

```
pax -rw -pe ./ /etc/mail
```

6. Change the current working directory to **/usr**. Issue:

```
cd /usr
```

7. Remove the files and directories under and including the **/usr/mail** directory.

```
rm -fr /usr/mail
```

8. Create a symbolic link for **/usr/mail** that points to **/etc/mail**.

```
ln -s /etc/mail /usr/mail
```

When you are done, you have customized the **mail** utility. If you unmounted any HFS data sets that were mounted on or below **/usr/mail**, you can mount them now using the same mount point as before.

Remounting a mounted file system

To remount the file system, use the TSO/E UNMOUNT command or the ISPF shell. The REMOUNT operand on the UNMOUNT command specifies that the specified file system be remounted, changing its mount mode.

Conditions under which you would remount a mounted file system are as follows:

- Maintenance cannot be performed on a read-only file system. The file system must be unmounted and then mounted again as read/write. If there are cascaded mounts, all of the file systems mounted on top of that file system must also be unmounted. You can unmount and remount a root file system. However, if the HFS is a shared read-only root HFS in a sysplex, you have to unmount the root on all other systems in the sysplex.
- When you are not using shared HFS, you can use the remount facility to mount file systems as read-only under normal operating situations and as read/write to perform maintenance.

You can use REMOUNT in shared HFS if all systems in the sysplex have REMOUNT support (are at least at V1R5 or have the V1R4 PTF installed). If there is at least one downlevel system, a remount request from an uplevel system will fail.

Another way to remount a file system in an environment where shared HFS is being used is to unmount the file system and then mount it again in the desired mode.

If a file is opened for a write, this is not checked if a remount operation changes the file system from read/write to read-only. Subsequent writes to the file will fail.

If a problem occurs with the remount, determine the failure, correct the problem, and try the remount again. The file system may be unavailable until the problems are corrected.

Remounting with zFS aggregates

You can remount zFS aggregates if the primary file system is mounted. It will succeed when remounting from R/O to R/W or from R/W to R/O because the aggregate is detached during the unmount and then is attached in the new mode during the mount.

However, remounts for HFS-compatible file systems in a sysplex will not work if both the clone and primary are mounted.

Copying the file system

To copy HFS data sets, use the DFSMSdss COPY DATASET command. For more information, see *z/OS DFSMSdss Storage Administration Reference*.

Backing up HFS data sets

Many new applications in z/OS exploit z/OS UNIX and store data in the HFS. Some customers have ported their applications over from other UNIX or NT platforms, but may not be familiar with the program products available to back up those applications on the z/OS platform. Other customers have used MVS and z/OS for a while, and assume that the backup techniques that they use for their MVS data sets are adequate for HFS files as well. This section addresses these issues and helps you plan your backup strategies for your HFS data.

Ways to back up HFS data sets

There are three models for backing up applications:

- The application maintains a transaction log for each transactional unit of work. A backup can be taken any time the application is running. If the data is recovered, the transaction log can be used to either back out, or redo, the transactions to reach a known sync-point for the application.
- The application provides a quiesce capability, during which time transactional updates are suspended. Either the application initiates this to perform its own backups, or an outside person or program product initiates this, takes the backup, then reactivates the application to full read/write mode when the backup has completed.
- The application provides neither of the above. In this simplest case, it is suggested that the backups occur only when the files are closed. If backups are taken while the files are open, it might be difficult to determine which transactions were fully recorded, and which others were still in flight.

IBM offers three ways to perform backups and maintain an inventory of their attributes:

- The z/OS DFSMSHsm feature
- The program product Tivoli Storage Manager, formerly known as ADSTAR Distributed Storage Manager (ADSM)
- The z/OS DFSMSDss feature

DFSMSHsm

Unlike other non-VSAM data sets that can be opened and closed repeatedly throughout the day, some HFS data sets are often mounted for several days or weeks at a time, with the individual file members inside opened as needed. Normally, DFSMSHsm's automatic backup (AUTOBACKUP) processes HFS data sets at most once per mount, so an HFS data set mounted for a week would only have one backup taken for that week. For some applications, that may not be frequent enough. Fortunately, DFSMSHsm™ provides some alternatives to ensure that backups are taken more frequently.

- An SMS-managed storage group can be defined with guaranteed backup frequency (GBF). For example, if GBF=3 days, then if a backup has not been taken for a particular data set in the last three days, a fresh backup is taken, whether the file has been updated or not. Since this applies to all data sets on a storage group, some customers have placed their HFS data sets into a unique storage group with a specification of GBF=1, so as not to affect other types of data.
- Backups once a day may not be frequent enough. DFSMSHsm provides commands to invoke backups to be taken, independent of the standard autobackup cycle and window. The BACKVOL TOTAL command can be used to back up all the files on a single DASD volume, a list of DASD volumes, a single storage group, or a list of storage groups. This command can be invoked from a job scheduling package such as OPC, or console automation package, such as Netview.
- If HFS data sets are intermixed on the DASD volumes with other data set types, you might want to back up the HFS data sets individually. You can use the DFSMSHsm command BACKDS to back up a single data set, or a set of data sets that match a particular mask filter. The DFSMSHsm batch program ARCINBAK can be used to back up a list of data sets that support JCL backward reference and variable substitution. DFSMSHsm also provides ABACKUP, which identifies which HFS data sets are part of a single aggregate list, and backs

these up as a single entity. You can invoke both the BACKDS and ABACKUP commands from job scheduling or console automation software.

- If the application was developed in-house, you can modify it to perform the backups internally. It may be able to perform its own quiesce process, or coordinate time stamps with its own transactional log. DFSMSHsm provides the ARCHBACK assembler macro interface.

If an HFS data set is mounted for read/write to a single MVS image, it can be only be backed up by DFSMSHsm from the MVS image that has it mounted. For automatic backup, you may need to designate host affinity by specifying a system name associated with AUTOBACKUP for each storage group. For command-initiated backups, you may need to ensure that the commands or batch jobs are issued to the correct MVS image.

If the file system being dumped by DFSMSHsm is currently mounted as read/write, then this file system can only be dumped from the system on which it is mounted. If the file system is mounted as read-only or is in a sysplex (mounted read-only or read/write), then it can be dumped from any system that has access to it.

If you use DFSMSHsm, you must define a user ID for the DFSMSHsm address space. For DFSMSHsm to access the HFS data sets, it must run under a user ID that is set up for access to a z/OS UNIX system. When you set this access up:

- The default group for the DFSMSHsm user ID must have an OMVS segment defined and a group ID associated with it.
- The home directory should be the root file system.

Tivoli Storage Manager

Tivoli Storage Manager offers another way to back up HFS data sets. You can use this program in combination with, or instead of, DFSMSHsm for your backup needs related to HFS data sets systems. Tivoli Storage Manager is a client/server based product and offers some additional features that are not available from DFSMSHsm.

- The z/OS UNIX System Services Client is available to back up the individual files and directories within a file system.
- Tivoli Storage Manager features a Central Scheduling component which can schedule z/OS UNIX client activity, such as “Selective Backup” at defined intervals. For example, you can back up the files in a specific subdirectory every four hours.
- Separate policies can be applied to a specific client node such that individual directories and files within a given file system can be effectively managed using retention, expiration and versioning attributes. These sophisticated features of Tivoli Storage Manager used in conjunction with a comprehensive INCLUDE/EXCLUDE list on the z/OS UNIX client platform provide a great deal of control over what is backed up and how the data is managed.
- End users can recover individual files that they have appropriate authority access to.

DFSMSdss

If you use DFSMSdss to dump or restore an active HFS data set, the user ID must be set up to have superuser authority to quiesce and unquiesce a file system. If HFS data sets are not mounted, then it is treated as an MVS data set and the user ID must have read authority for dump purposes and update authority for restore purposes.

Creating the user file systems

A user HFS data set is allocated in exactly the same way as you created the root HFS data set. Choose a data set name that has the user name as one of the qualifiers and a size that provides sufficient space for the user's requirements.

Although the HFS data set does not have to be SMS-managed, it is still highly recommended. Multivolume HFS data sets are only supported as SMS-managed. (That is, you cannot have multivolume non-SMS-managed data sets.) As a user adds files and extends existing files, the data set increases in size to a maximum of 123 extents if secondary extents are specified in the allocation.

If more space is required, you may wish to increase the size on the allocation or you may wish to create additional HFS data sets on different DASD volumes for a user and mount them at different mount points in the user's hierarchy.

The newly allocated data set has a root whose permission bits are set at 700. You can change the permissions only after the data set is mounted. See "Changing the permission bits for a file" on page 92 for more information on changing permission bits for a file or directory.

The sample JCL to create an HFS data set is in Figure 12. Change the JCL where needed.

```
//STEP01 EXEC PGM=IEFBR14
//HFS      DD DSN=OMVS.USER1,SPACE=(CYL,(10,1,1)),
//          DSNTYPE=HFS,DCB=(DSORG=P0),
//          DISP=(NEW,CATLG,DELETE),
//          STORCLAS=OPENMVS
```

Figure 12. JCL to allocate user HFS data sets

Making user file systems available

After the user's HFS data set is allocated, you need to get it mounted at a mount point off the root directory to make it available. The preferred place to mount all user HFS data sets is a user directory under the **/u** user directory. In z/OS, there are two ways to accomplish this:

1. Direct mount. For a direct mount, allocate an intermediate HFS data set (we called it OMVS.USERS) to be mounted between the root file system and all user file systems. Create a mount point using the **mkdir** command and issue the **mount** command. (To make the mount permanent you will also need to add the HFS data set name and its mount point to the BPXPRMxx member of parmlib.) Figure 13 on page 130 shows this.

For more information, see "Using direct mount" on page 131.

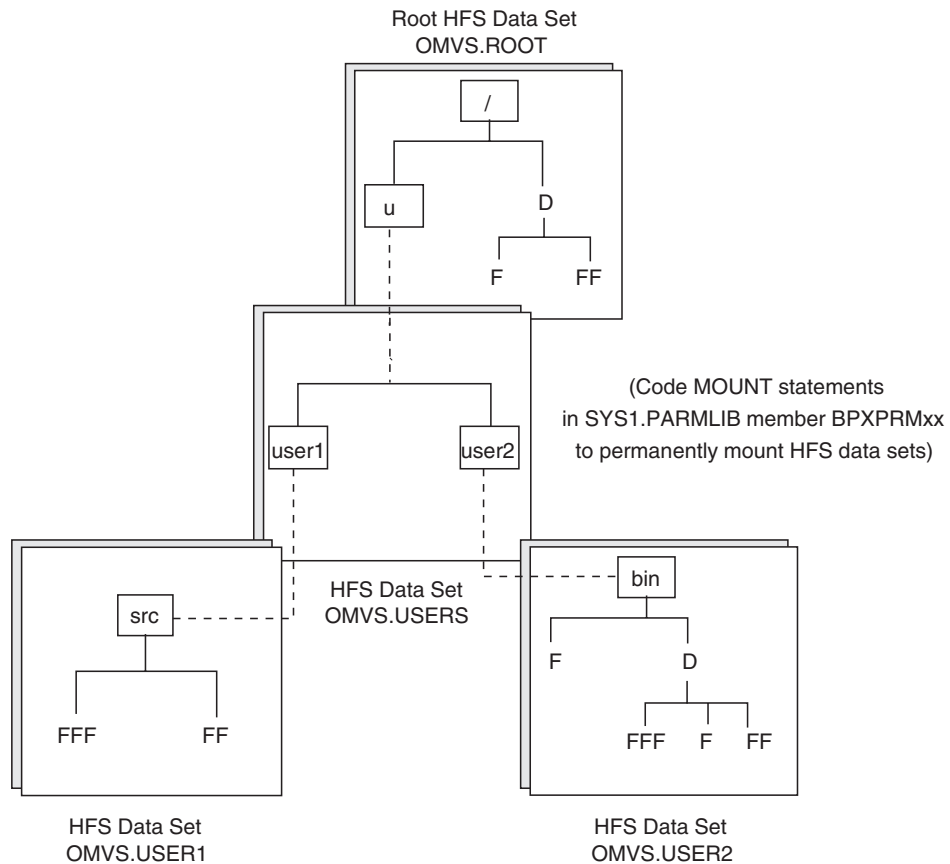


Figure 13. Direct mount

2. Automount facility. You will have 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. Figure 14 on page 131 shows this. See “/etc/auto.master” on page 149 for more information.

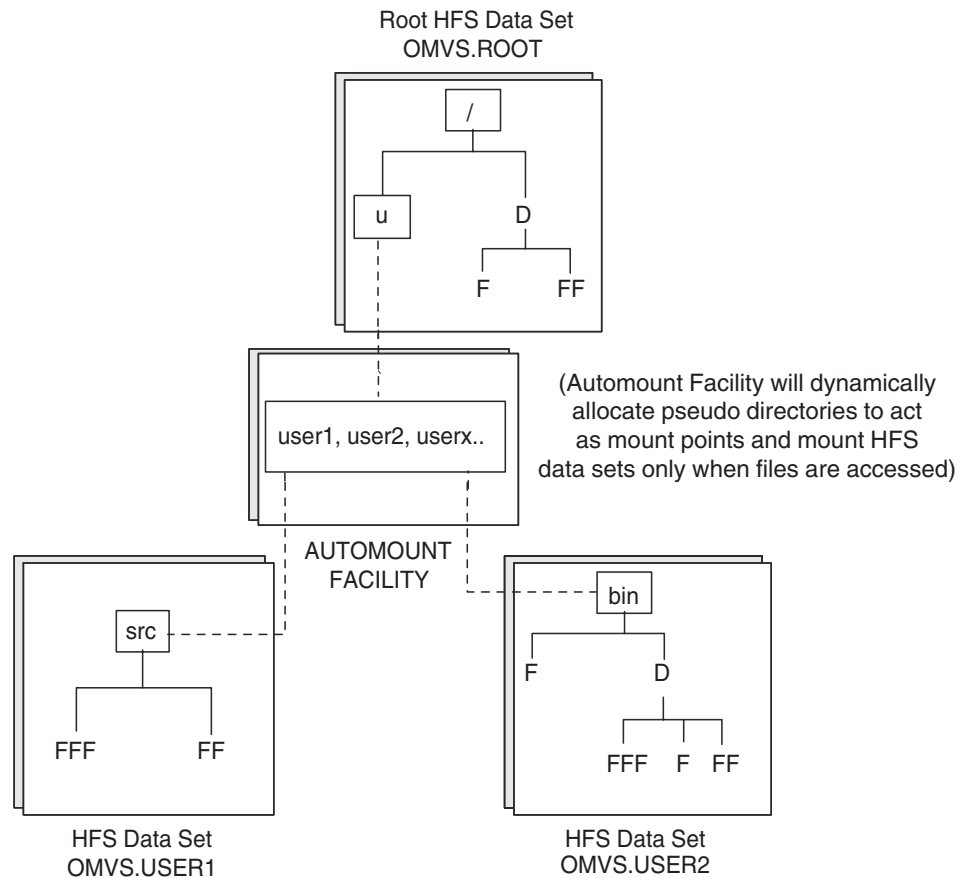


Figure 14. Automount facility

Using direct mount

The root file system should be set up so that it does not require frequent changes or updates (outside of SMP/E maintenance). To achieve this we will allocate an intermediate HFS data set called OMVS.USERS and mount it at **/u**.

All user directories that are added will reside in this new HFS data set and not in the root HFS data set. Sample JCL to allocate this intermediate HFS is in Figure 15. Change the JCL to fit your environment.

```
//STEP01 EXEC PGM=IEFBR14
//HFS      DD DSN=OMVS.USERS,SPACE=(CYL,(5,1,1)),
//          DSNTYPE=HFS,DCB=(DSORG=PO),
//          DISP=(NEW,CATLG,DELETE),
//          STORCLAS=OPENMVS
```

Figure 15. JCL to allocate intermediate HFS data set

The next thing to do is mount this new intermediate HFS data set at **/u**. The mount can be performed from an ID that has superuser authority by:

- Using the **usr/sbin/mount** REXX exec from the shell
- Using the TSO MOUNT command
- Using the **mount** shell command

- Using the ISHELL File_Systems pull-down
- Adding an entry to the BPXPRMxx member in SYS1.PARMLIB so that it will be mounted when the system re-IPLs.

An example of the commands required, including issuing the **mount** command from the shell is shown in Figure 16. Type OMVS from ISPF option 6 to enter the shell. Then execute the highlighted commands to mount the HFS data set OMVS.USERS. In this example, the user ID with superuser authority called ADMIN is used.

```

IBM
Licensed Material - Property of IBM
5655-068 (C) Copyright IBM Corp. 1993, 1995
(C) Copyright Mortice Kern Systems, Inc., 1985, 1994
(C) Copyright Software Development Group, Univ. of Waterloo, 1989

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 -
- - - - -

# /usr/sbin/mount /u omvs.users      1
OMVS.USERS is now mounted at
/u
# df -P                              2
Filesystem      512-blocks      Used Available Capacity Mounted
OMVS.USERS      7200              40      7160      1% /u
OMVS.ROOT       82800            79608     3192     97% /
# chmod 755 /u      3

===>

ESC=¢ 1=Help  2=SubCmd 3=HlpRetrn 4=Top      5=Bottom 6=TSO
      7=BackScr 8=Scroll 9=NextSess 10=Refresh 11=FwdRetr 12=Retrieve

```

Figure 16. Mounting the new intermediate HFS data set

- 1 Use the **mount** command to mount the HFS data set, OMVS.USERS, on mount point **/u**.
- 2 Run the display free space command to display the mounted file systems.
- 3 Change the permission bits to allow access to **/u**.

Now that the OMVS.USERS HFS data set is mounted at mount point **/u** you can create the user1 mount point from a superuser ID by using:

- The **mkdir** command in the shell
- The TSO/E MKDIR command
- The ISHELL Directory pull-down

Figure 17 on page 133 shows the sequence of commands performed by a superuser in the shell to create a mount point for a new user off **/u**. Type in OMVS from ISPF option 6 to enter the shell and execute the highlighted commands to create the user1 mount point. Remember, before creating a mount point directory for a new user, the new user must be defined to your security product OMVS segment.

```

# cd /u
# pwd
/u
# mkdir -m 700 user1
#ls -l
total 16
drwx-----  2 ADMIN  OMVSGRP          0 Nov  7 09:07 user1
#
===>

ESC=¢ 1=Help    2=SubCmd 3=HlpRetrn  4=Top      5=Bottom  6=TSO
      7=BackScr 8=Scroll 9=NextSess 10=Refresh 11=FwdRetr 12=Retrieve

```

Figure 17. Creating a user's mount point directory

- 1 Change to make **/u** your current working directory.
- 2 Check to make sure **/u** is the current working directory.
- 3 **/u** is the current working directory.
- 4 Create a new directory for user1 setting the permission bits to 700. See "Controlling access to files and directories" on page 89 for information on permission bit settings.
- 5 List the contents of the **/u** directory.
- 6 The **user1** directory entry.

The user HFS that was previously created can now be mounted at **/u/user1**. The mount can be performed by:

- Using the **/usr/sbin/mount** REXX exec from the shell
- Using the TSO/E MOUNT command
- Using the ISHELL File_systems pull-down
- Adding an entry to the BPXPRMxx member in SYS1.PARMLIB so it will be remounted when the system re-IPLs.

Figure 18 on page 134 shows an example of the commands required, including issuing the **mount** command from the shell. Type in OMVS from ISPF option 6 to enter the shell and execute the highlighted commands to mount the HFS data set OMVS.USER1.

```

# /usr/sbin/mount /u/user1 omvs.user1 1
OMVS.USER1 is now mounted at
/u/user1
# df -P 2
Filesystem      512-blocks      Used Available Capacity Mounted
OMVS.USER1      12960            40      12920      1% /u/user1
OMVS.ROOT       82800          79608       3192      97% /
# chown user1:grpoe /u/user1 3
# ls -l /u/user1 4
total 16
drwx-----  2 USER1  GRPOE      0 Nov  7 09:09 user1
#

===>

ESC=¢ 1=Help    2=SubCmd 3=HlpRetrn 4=Top      5=Bottom  6=TSO
      7=BackScr 8=Scroll 9=NextSess 10=Refresh 11=FwdRetr 12=Retrieve

```

Figure 18. Mounting the new HFS data set

- 1 Issue the **mount** command to mount the HFS data set, OMVS.USER1, on mount point **/u/user1**.
- 2 Run the display free space command to display the mounted file systems.
- 3 In order for USER1 to use this new file system, you must issue the **chown** command to change the ownership and to change the group to the user's default group. Issue this command to set the owner and group fields of this mount point directory for the USER1 ID. You only need to issue the **chown** command once because the values will be saved in the new file system and will be reused even when the file system is remounted later.
- 4 Issue a list command to display the new directory for USER1.

If you want to make the mounting of the OMVS.USERS and OMVS.USER1 HFS data sets permanent, you have to add an entry in the BPXPRMxx member of SYS1.PARMLIB. These two mount statements should follow the ROOT statement for the root file system.

```

MOUNT    FILESYSTEM('OMVS.USERS')
          TYPE(HFS)
          MOUNTPPOINT('/u')
          MODE(RDWR)

MOUNT    FILESYSTEM('OMVS.USER1')
          TYPE(HFS)
          MOUNTPPOINT('/u/user1')
          MODE(RDWR)

```

Using file locks

Programs using files can specify locks. Locks are used to lock byte ranges in files and are intended for use by cooperating application programs.

For systems participating in shared HFS, there is only one byte range lock manager. For more information, see “Locking files in the sysplex” on page 182.

Programs can use file locks via the `fcntl()` function. See *z/OS UNIX System Services Programming: Assembler Callable Services Reference* and *z/OS C/C++ Run-Time Library Reference* for more information about `fcntl()`.

Locks are advisory. (Advisory locking is used in UNIX systems.) Consequently, more than one program can update a file at the same time. Keep advisory locking in mind during problem determination.

Creating special files

There are several types of special files:

- A **character special file** is a file that provides access to an input/output device. Examples of character special files are: a terminal file, a NULL file, a file descriptor file, or a system console file. Each character special file has a device major number, which identifies the device type, and a device minor number, which identifies a specific device of a given device type. Character special files are customarily defined in **/dev**; these files are defined with the **mknod** command. You must have UID(0) to create a character special file. The best way to obtain UID(0) is to be defined to BPX.SUPERUSER FACILITY class. Then issue the **su** command to switch to UID(0) before issuing the **mknod** command. You cannot share character-special files in read/write mode among systems participating in shared HFS in a sysplex.

- A **pipe** is a way to communicate in first-in-first-out (FIFO) order from one or more processes to one or more processes. Pipes are treated as though they were files.

A pipe sends data from one process to another or back to itself. By forking processes, a pipe can be shared by a number of processes—for example, written to by three processes and read by seven.

A program creates a pipe with a `pipe()` function. The pipe vanishes when the last process closes it. A pipe does not have a name in the file system; a pipe is also called an *unnamed pipe*.

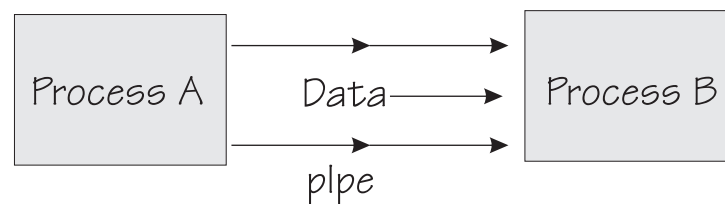


Figure 19. A pipe between two processes

- A **FIFO special file** sends data from one process to another so that the receiving process reads the data first-in-first-out (FIFO). A FIFO special file is also called a *named pipe*, or a *FIFO*. A FIFO special file can also be shared by a number of processes that were not created by forks. A FIFO special file can be written into and read by the same process using multiple threads.

FIFO special files can be shared between systems exploiting shared HFS. For more information on shared HFS, see Chapter 9.

A program creates a FIFO special file with a **mkfifo** command or a `mkfifo()` function. The name is maintained in the file system until the named pipe is deleted by an **rm** command or an `unlink()` function.

- A **UNIX domain socket address file** represents socket addresses in the UNIX domain.

These files cannot be shared in read/write mode among systems participating in shared HFS in a sysplex.

To prepare for using AF_UNIX (local) sockets, the AF_UNIX physical file system (PFS) creates a socket address file in the file hierarchy during the `bind()` function

call. The files are defined as specified by the program that calls `bind()` and are usually in the user's home directory, the root directory, or in `/tmp`.

Because they are part of the file system, be careful not to delete any of these socket address files by accident. If you do delete them, programs will not be able to connect to or send datagrams to the program that created the file.

Pipes and FIFO special files are created by programs and users; character special files are typically created by the system programmer.

Pseudoterminal files

Pseudoterminals (pseudo-TTYs) are used by users and applications to gain access to the shell. A `pseudo_TTY` is a pair of character special files, a 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)

The `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 or logs in using `rlogin` or `telnet` to initialize a shell, the system selects an available pair of these files. The pair represents the connection. The maximum number of pairs is 10000. You can specify an appropriate number of pairs in the `MAXPTYs` parameter; see "MAXPTYs" on page 33.

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.

Pseudo-TTY files are dynamically created by the system when they are first referenced. You can add pseudo-TTYs with `MKNOD` TSO/E commands, such as in Figure 20, or with **`mknod`** shell commands. You can also use the ISPF shell to perform these functions.

```
MKNOD '/dev/ptyp0000' major(1) minor(0) mode(6 6 6)
MKNOD '/dev/ttyp0000' major(2) minor(0) mode(6 6 6)
```

Figure 20. MKNOD TSO/E commands for a master-and-slave pair

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 entered directly in a TSO/E session or a shell session.

Null file

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.

File descriptor files

A file descriptor file, `/dev/fd/n` or `/def/fd/n` (major 5, minor *n*) is used to refer to the same file as a previously opened file, as indicated by file descriptor *n*. If file *n* is a regular file or a character special file, the open for `/dev/fd/n` or `dev/fd/n` will be done as a real open of the file with file descriptor *n*. Otherwise, the dup protocol will be used for that open.

When naming file descriptor file, the *n* in `/dev/fd/n` or `/dev/fd/n` is the same as the minor number. The minor number determines which file descriptor number to duplicate. For example, opening `/dev/fd/1` creates a file descriptor that is a duplicate of file descriptor 1. This might be useful for a program that expects a file name for output, but you may want it to write its output to stdout instead.

`/dev/fd/n` files are used by **c89** to avoid the name-length limitations imposed by the DD statement PATH parameter.

Use of **c89** assumes that you follow the naming conventions for file descriptor files.

File descriptor files are created dynamically as needed by the system when they are first referenced.

UNIX domain socket name special file

A pathname specifies the socket address for a UNIX domain socket. The pathname is assigned by the application programmer. There is no convention for the name. The operating system creates the file (major 6).

System console file

Data written to this file is sent to the console and displayed using a write-to-operator (WTO). An example of a system console file is `/dev/console` (major 9). It is automatically created the first time the system is IPLed.

Handling file system failures

If the file system fails, the operator must take several steps to restore it. See “File system failure” on page 272 for this information.

Restoring the root file system

If the physical file system owning the root fails, all work in progress when a failure occurs is lost, and it must be restarted from the beginning.

CAUTION:

Unmounting and remounting a root file system is very disruptive to the system. Any work in progress must be undubbed and redubbed.

Rule: The person who restores a failed root file system or an unmounted root file system must be a superuser who is defined with a home directory of / (root).

Recovering from file system problems with the root

If the root file system becomes corrupted, you can restore to the last known good copy and IPL the system, or you can avoid doing an IPL by following the steps described in this section.

This procedure, while it does not require an IPL, is disruptive to all UNIX processes. For example, any work that depends on TCP/IP will be affected.

Steps for recovering from file system problems with the root

Before you begin: You need to have a terminal that does not depend on TCP/IP, and the user ID doing the unmounts must be defined as UID(0) or have appropriate privileges under UNIXPRIV.

Perform the following steps to recover from file system problems with the root.

1. List the applications that are running.

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

2. Bring down all the processes listed, except for BPXOINIT. Issue D OMVS,A=ALL again to make sure all the processes except BPXOINIT have been shut down. The display will look like the following:

```
BPX0040I 07.31.14 DISPLAY OMVS 017
OMVS      000E ACTIVE          OMVS=(65)
USER      JOBNAME  ASID      PID      PPID STATE      START      CT_SECS
IBMUSER  BPXOINIT   0013        1        0 MR---B    07.21.27    .034
LATCHWAITPID=      0 CMD=BPXPINPR
SERVER=Init Process      AF=      0 MF=000000 TYPE=FILE
```

3. Identify the file systems that are mounted. Issue:

```
D OMVS.F
```

You will see a display that shows the mounted file systems.

4. Unmount the file systems by using the TSO/E ISHELL command.

Rule: The root file system must be unmounted last and you must use the IMMEDIATE option when unmounting the root file system. After the root has been unmounted, the mount table should show SYSROOT.

5. Issue

```
D OMVS, F
```

You should see a display similar to the following:

```
BPX0044I 10.38.16 DISPLAY OMVS 054
OMVS      000E ACTIVE          OMVS=(65)
TYPENAME  DEVICE  -----STATUS-----  MODE QJOBNAME  QPID
BPXFTCLN   0 ACTIVE                      RDWR
NAME=SYSROOT
PATH=
```

6. Follow your recovery actions for the root file system.

7. Mount the root. From TSO READY or ISPF Option 6, issue:

```
MOUNT FILESYSTEM('your root dsname') TYPE(HFS) MOUNTPPOINT('/')
```

The root is then mounted at / in read/write mode. If your root is read-only, add MODE(READ) to the above command.

-
8. Mount the individual file systems at their respective mount points by using the TSO/E ISHELL command.

Tip: View the output from the first D OMVS,F command, or the BPXPRMxx member that you IPL with to determine the mount points.

-
9. After the file systems are mounted (check by issuing D OMVS,F again), have a superuser (UID=0) enter the OMVS shell and issue **/etc/rc** from the prompt to run the shell initialization script.

-
10. Follow your procedures to restart other applications (such as TCP/IP, NFS, FTP, and WebServer) and confirm that all functions are working.
-

You know you are done when you have rebuilt the z/OS UNIX environment and all functions are working.

Installing service into the HFS

Some customers install service in response to a particular problem that they experience; some customers install service to prevent problems from occurring. This latter method is called “preventive service.” In either case, service is installed in the HFS in a way similar to how it is installed into MVS data sets. This section describes how to install service into the HFS.

To install service, system programmers create a copy of the system that they are migrating from onto another pack. Sometimes the system that they are migrating from is the active production-level system, called the *driving system*. This new copy is called the *target system*. The DDDEFs or the DD statements in the cataloged procedure that is used when applying service are updated to point to the libraries on the target system. When service is applied, updates are made to the target libraries.

After service is installed, the new target libraries are tested, and if successful, are put into production as the new driving system.

As you prepare to install service into the HFS, keep the following in mind:

- There is only one file hierarchy active at any given time. You may have multiple HFS data sets on your system. But z/OS UNIX does not recognize them unless they are mounted at a directory (mount point) within the file hierarchy.
- If you install service directly on the production file system, you will copy new load modules over existing ones. This causes potential tracking and system-level problems. Therefore, you should create a copy of the production file system before installing service.

In most cases, you will copy the root file system. However, you can use this same concept to duplicate other production HFS data sets that are mounted in the file hierarchy or in individual directories. For help copying the root file system, see “Copying the file system” on page 126.

This new copy must be mounted at a directory (mount point) within the active file hierarchy. The directories in the newly mounted file system will be the target libraries when installing service.

- The distribution libraries for elements installing into the HFS are still partitioned data sets.

Installing service into the HFS involves the following steps. In these steps, the new HFS data set is called the *service HFS data set*. The first two steps are shown in Figure 21 on page 141.

1. Create a clone of the system that you are migrating from. This includes copying all necessary partitioned data sets and HFS data sets. A number of utilities such as IEBCOPY or DFSMSDss can be used to copy partitioned data sets. For details on copying HFS data sets, see *z/OS and z/OS.e Planning for Installation*.
2. With a superuser ID, mount the service HFS data set at a mount point within the active file hierarchy. To do this, first create a directory (mount point) for the **/service** directory:
 - a. Issue the TSO MKDIR command and create a directory called **/service**:

```
mkdir '/service'
```
 - b. Issue the TSO/E MOUNT command to mount the service HFS data set to the root file system.
 - c. At this time, the **/service** directory has permission bits of 755. This prevents unauthorized users from corrupting the service HFS.
3. Change the DDDEFs or DD statements used by the SMP/E cataloged procedure to point to the new target directories. For example, the DDDEFs must now point to **/service/bin/IBM** instead of to **/bin/IBM**.

Note: SMP/E allows you to perform the ZONEEDIT function for all directories. You no longer need to change individual DDDEFs for directories manually.

Also change the VOLSER information of the DDDEFs or DD statements for the partitioned data sets.

4. Install the service.
5. Test out the new target libraries.
6. After the target libraries have been successfully tested, you can move them into production. To replace the original HFS data set with the service HFS data set, using either one of the following methods:
 - Use DFSMSDss DUMP and RESTORE to copy the service HFS data set to the original HFS data set.

Or

- Unmount the service HFS data set from **/service** and mount it on the original HFS data set mount point. This step might require changes to the BPXPRMxx parmlib member. A re-IPL is also required.
7. Keep the target system SYSRES and the target system HFS synchronized, because service could affect both files in the HFS and members of the partitioned data set. Make both the target system SYSRES and the target system HFS available at the same time.

Example of installing service

Figure 21 shows a target system that was created by making a copy of the driving system on another pack and by making a copy of the root file system. The root file system is copied into another HFS data set and is mounted within the file hierarchy. In this example, it is mounted to the **/service** directory.

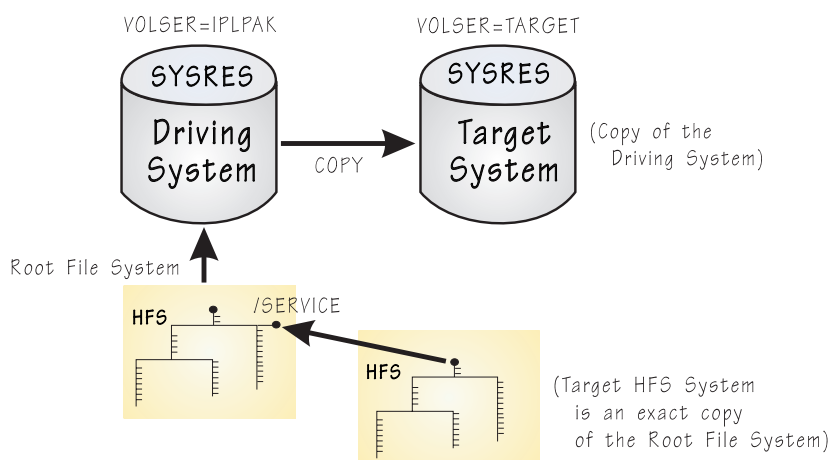


Figure 21. Preparation for installing service

The DDDEFs or the DD statements used by the cataloged procedure when applying service must point to the target system.

Example: The following example shows sample DDDEFs pointing to new target libraries:

```
LPALIB DD DSN=SYS1.LPALIB,VOLSER=TARGET,  
UNIT=3390,DISP=SHR
```

```
SFSUMBIN DD PATH='/service/bin/IBM'
```

Transporting the HFS from the driving system to the target system

Some MVS installations install products and service on one system and then transport this system image to the rest of their enterprise. Using the DFSMSdss DUMP and RESTORE utilities, you can dump individual product libraries or full volumes, transport them to other systems, and restore them.

However, because the individual HFS data sets that make up the active file hierarchy might be on SMS-managed volumes, there are some special considerations for making a transportable copy:

- You can dump each HFS data set to be transported into individual sequential data sets using the DFSMSdss dump utility. These sequential data sets contain all the necessary information about the files and can also exist with other product libraries that need to be transported.
- After the system image has been transported to the target system, you can restore individual product libraries or full volumes using the DFSMSdss restore utility.

- After the data sets have been unloaded on the target system, you can use the DFSMSdss restore utility to restore the sequential data sets into individual HFS data sets. These HFS data sets will make up the active file hierarchy on the target system.

Using the process just outlined, you can duplicate system images across the enterprise.

This process is also known as using a “one-pack system.” A one-pack system consists of one logical SYSRES volume.

See the following for information on how to dump and restore a hierarchical data set:

- *z/OS DFSMS Migration*
- *z/OS DFSMSHsm Storage Administration Guide*
- *z/OS DFSMSdss Storage Administration Reference*

Installing service into /etc

Starting in OS/390 V2R9, **/etc** is a symbolic link, not a directory as in prior releases. This change affects how you install service into **/etc**. Use the procedures in this section if the installation process must create one or more **/etc** directories.

Because it is a symbolic link, **/etc** cannot receive product or service code. If you are installing service or products that must write to **/etc**, you need to change **/etc** to a directory, install the code, and then change **/etc** back to a symbolic link. The steps for those tasks are in SYS1.SAMPLIB; they are:

1. Mount a clone of the HFS you are installing into the **/SERVICE** mount point.
2. Run the sample job BPXISSTD to convert the **/SERVICE/etc** symbolic link to a directory. Pass the **/SERVICE** parameter to the REXX exec.
Optionally, you can use BPXISJCL to submit the job in the background.
3. Mount the clone of the **serviceETC.HFS** of the system you need to service at **/SERVICE/etc**.
4. Install the service, which may include running REXX execs that create directories under **/SERVICE/etc**.
5. Unmount the **/etc** after everything is installed from **/SERVICE/etc**.
6. Run the sample job BPXISSETS to convert the **/etc** back to a symbolic link at **/service/etc**.

Installing products into the HFS

When you install other products into the file system, create new directories where the files associated with the new product will be installed. You may also need to create a new HFS data set for the new product and mount it to a new directory.

To help you decide how many HFS data sets you need, see the information about product sets in the section that discusses placing data sets on specific volumes in *z/OS and z/OS.e Planning for Installation*.

The procedures in this section apply to those installing products into the **/etc** file system on a production system.

Beginning in OS/390 V2R9, **/etc** is a symbolic link. It cannot receive product or service code. You need to run jobs that change **/etc** to a directory, install the code, and change **/etc** back to a symbolic link. The jobs you use are in SYS1.SAMPLIB.

Here are the steps to follow:

1. Mount a clone of the root HFS at the **/service** mount point.
2. Run the sample job BPXISSTD to convert the **/etc** symbolic link to a directory.
Pass the **/etc** as a parameter to the REXX exec.
Optionally, you can use BPXISJCL to submit the job in the background.
3. Mount the **/etc** file system.
4. Install the products or service.
5. Unmount the **/etc** file system after everything is installed.
6. Run the sample job BPXISST to convert the **/etc** back to a symbolic link.

You need to follow these steps each time you install code into **/etc**.

Chapter 7. Using the zSeries File System (zFS)

Overview

This chapter discusses the zSeries File System (zFS) component of the IBM z/OS Distributed File Service product. zFS is a UNIX file system that can be used in addition to HFS. It contains files and directories that can be accessed with APIs. They can also be mounted into the z/OS UNIX hierarchy along with other local or remote file systems types such as HFS, TFS, and NFS.

You can use ISPF panels to create and manage the zFS file system. For more information about working in the ISPF shell, see *z/OS UNIX System Services User's Guide*.

Both HFS and zFS can be automounted. For information about zFS and the automount facility, see "Automounting zFS file systems" on page 147.

Restriction: For non-HFS compatible aggregates, the aggregates must be attached.

Rule: HFS is still required for z/OS installation and the root file system must be HFS.

For more information about setting up and administering zFS, see *z/OS Distributed File Service zSeries File System Administration*.

How does zFS differ from HFS?

zFS and HFS are both UNIX file systems and both can participate in shared sysplexes. However, while HFS always has a single file system per data set, zFS may have multiple file systems in a single data set. These data sets are called "aggregates" and are a collection of data sets.

When would you want to use zFS?

zFS provides the following features and benefits:

- Performance gains in many customer environments when accessing files approaching 8K in size that are frequently accessed and updated. The access performance of smaller files is equivalent to that of HFS.
- Space sharing. Multiple zFS file systems can be defined in a single data set, enabling space that becomes available from erasing files in one file system to be available to other file systems in the same data set. This is an optional function that is available only in a nonsysplex environment.
- Read-only cloning of a file system in the same data set. The cloned file system can be made available to users to provide a read-only point-in-time copy of a file system. This is an optional feature that is available only in a nonsysplex environment.

zFS and shared sysplex

Because zFS can be used in a sysplex, users in a sysplex can access zFS data that is owned by another system in the sysplex. zFS file systems will be automounted if there is a system failure, and can be automounted.

Rule: For full support, zFS must be running on all systems in the sysplex. For a complete list of restrictions, see *z/OS Distributed File Service zSeries File System Administration*.

See *z/OS Planning for Multilevel Security* for more information about shared HFS and system-specific security labels.

Mounting considerations

For zFS, only one file system may be defined inside the aggregate, and the file system's name must be the same as the aggregate name. This enables these file systems to be mounted without being attached because zFS can easily determine the data set name from the file system name. This is referred to as "HFS compatibility mode".

An HFS-compatible file system can be cloned; in this sense, these aggregates may contain two file system, but the clone file system exists with certain restrictions that make it easier to handle. That is, only one clone is allowed at a time, and the name of the clone is the same as that of the file system, with **.bak** added on at the end. An HFS-compatible file system and its clone, if both are mounted, must still be physically mounted from the same sysplex system, but the mount commands do not have to be entered from the same system.

Mount behaviors

This section describes the mount behaviors of zFS.

- You can mount an HFS-compatible zFS file system or its clone without attaching its aggregate.
- You can mount either of these file systems.
- They can be mounted from different sysplex systems without the SYSNAME() operand. The second one that is mounted will become owned by the system that owns the first. If the second one is mounted with SYSNAME(), then that name must be the correct name of the system where the first one was mounted.
- The AUTOMOVE operand of file systems in an HFS-compatible aggregate may be different. If their SYSLISTs are incompatible, the mounts will be accepted but later automoves may fail when those SYSLISTs have to be used.
- Non-HFS compatible zFS file systems can only be mounted after their aggregate has been attached and only from the system where the attach was done.
- If an HFS-compatible aggregate is explicitly attached on any system, then its file systems can only be mounted from that system.

Read-only BPAM access

You can set up read-only basic partitioned access method (BPAM) access to zFS files. Each zFS directory is treated as if it were a PDSE or PDS directory. For more information about BPAM, see *z/OS DFSMS: Using Data Sets*.

Chapter 8. Using the automount facility

Overview

The automount facility automatically mounts file systems at the time they are accessed.

Using the automount facility provides the following advantages:

- Management of file systems is easier. You do not need to mount most file systems at initialization. You also do not need to request that operators perform mounts for other file systems. It is easier to add new users because you can keep your parmlib specification stable. You can establish a simple automount policy to manage user home directories.
- Resources are not consumed until they are requested. A file system that is managed by the automount facility remains unmounted until its mount point is accessed.
- You can reclaim system resources used by a mount if that file system has not been used for a period of time. You can specify how long the file system should remain mounted after its last use.

This chapter assumes that you are automounting HFS file systems, although you can use the automount facility to mount other file system types such as NFS and zFS file systems.

Restriction: If you are using system-specific security labels, do not use automount. For more information, see *z/OS Planning for Multilevel Security*.

In this chapter

Subtask	Associated procedure (see . . .)
Setting up the automount facility	"Steps for setting up the automount facility" on page 150

Automounting NFS file systems

You can use the automount facility for NFS. For information about the parameter requirements for NFS client mounts, see:

- *z/OS Network File System User's Guide*
- *z/OS Network File System Customization and Operation*

Automounting zFS file systems

You can use the automount facility for zFS.

Restrictions: For non-HFS-compatible aggregates, the aggregates must be attached. Since this assumes HFS-compatible zFS file systems, do not use this support if file systems in general aggregates are mounted with the same automount policy. If you do, unused data sets may be created. For more details, see *z/OS Distributed File Service zSeries File System Administration*.

Rule: Because this assumes zFS file systems that are compatible with HFS file systems, do not use the automount facility if file systems in general aggregates are

mounted with the same automount policy. If you do so, unused data sets may be created if the automount policy specifies that file systems should be created (via **allocany** or **allocuser**).

Automounting in a sysplex

In a sysplex environment, the most recent automount policy that was loaded prevails for all the systems participating in the sysplex.

Rules:

1. Keep automount policies consistent across all the systems in the sysplex.
2. Do not use the default automount delay time of 0. Instead, specify a delay time of at least 10.

How does the automount facility work?

You create an automount policy that specifies directories containing only mount points, which is the recommended method of managing user HFS data sets. As each mount point is accessed, an appropriate file system is mounted. The mount point directories are created as they are required. If the file system is no longer used, the mount point directories are deleted.

Think of the automount facility as an administrator that has total control over a directory. When a name is accessed in this directory, the automount facility checks the automount policy for the file system that is supposed to be associated with that name. Then it performs a **mkdir** followed by a mount and moves out of the way. Now the root directory of that newly mounted file system can be accessed as that name.

For example, suppose that you had created the USER1 directory with the **mkdir** command. If you had set up the automount facility and put the automount policy in place, you would not have needed to do that. The USER1 directory would have been dynamically created and the OMVS.USER1 data set automatically mounted at the **/u/user1** mount point.

Later, if the **/u/user1** file system was not accessed based on certain criteria in your automount policy, the OMVS.USER1 data set is automatically unmounted and the USER1 directory removed.

The automount facility will not manage any directory until it can process the entire policy without encountering any errors.

Using the automount facility

This section contains background information for the procedure described in “Steps for setting up the automount facility” on page 150.

You need to customize the configuration files before you can start using the automount facility. The **/etc/auto/master** file must contain the directory or directories that the automount facility will monitor. The **MyMap** file must have entries for the mapping between the subdirectory of a directory managed by the automount facility and the mount parameters. For an example of a **MyMap** file, see “MapName” on page 149.

You can also refer to the **automount** section in *z/OS UNIX System Services Command Reference* for more information about both files.

/etc/auto.master

The default file name of the master file is **/etc/auto.master**. It contains the directory or directories that the automount facility will monitor. It also contains an associated **MapName** file that contains the mount parameters.

Figure 22 is an example of a **/etc/auto.master** file. It indicates that the automount facility is to manage the **/u** directory. If someone using kernel services tries to access a directory in the **/u** directory, the automount facility automatically mounts the data set based on the example MapName policy in Figure 23.

/u	/etc/u.map

Figure 22. Example of an **/etc/auto.master** file, **/etc/u.map**

MapName

The **MapName** file contains the mapping between a subdirectory of a directory managed by the automount facility and the mount parameters. It can contain both specific entries and a generic entry. When the automount facility tries to resolve a lookup request, it attempts to find a specific entry. If a specific entry does not exist for the name being looked up, it will then attempt to use the generic entry.

Rule: The **MapName** file can contain only one generic entry, and it has to be the first entry in the **MapName** file.

Example: Following is an example of a **MapName** file. It contains the mount parameters for the user directories.

name	*
type	HFS
filesystem	OMVS.<uc_name>
mode	rdwr
duration	nolimit
delay	10

Figure 23. Example of a generic entry in a MapName file, **/etc/u/map**

When using generic entries, you should have a consistent HFS data set naming convention. The HFS data sets in the example have a high-level qualifier of OMVS, and the lower level qualifier is equal to the user ID.

In the example, **<uc_name>** specifies that the name being looked up is to be represented in uppercase. The automount facility creates a directory containing that name and uses it as a mount point for the file system to be mounted. You can use **<uc_name>** to replace any level qualifier in the data set. For example, if the name of the directory that is being looked up is USER1, the automount facility will resolve the name in the following ways:

OMVS. <uc_name> = OMVS.USER1
OMVS. <uc_name>.HFS = OMVS.USER1.HFS

You can also use the *lowercase* keyword to specify that the generic match is to be done on lowercase names only. Numbers and special characters are allowed.

Using map files from other systems

If you have been using a map file on another system and want to use it on z/OS UNIX, you will have to use a conversion utility to reformat it. If you want to use a map file with syntax different from that supported by z/OS UNIX, such as a map file from another type of system, you can implement a conversion utility that converts the file to syntax supported by z/OS UNIX. Then you can direct the automount utility to run it and use the output of your conversion utility as the map file. For more information, see the **automount** command in *z/OS UNIX System Services Command Reference*.

Steps for setting up the automount facility

Before you begin: You must have a superuser ID in order to activate the automount facility from the shell.

Perform the following steps to set up the automount facility.

1. Add the following statement to your BPXPRMxx parmlib member.

```
FILESYSTYPE TYPE(AUTOMNT) ENTRYPOINT(BPXTAMD)
```

2. Either restart OMVS or issue SETOMVS RESET to activate the automount PFS.

3. Customize the **/etc/auto.master** file. For more information about the **/etc/auto.master** file, see “/etc/auto.master” on page 149 and *z/OS UNIX System Services Command Reference*.

Recommendation: You should set the permission bits so that the file is protected from write by ordinary users like other system files. The files should be owned by UID(0) and have write permission only for owner, such as 644. If the group for the file is a properly restricted group, 664 would also be appropriate.

4. Customize the MapName file. For more information about the **MapName** file, see “MapName” on page 149 and *z/OS UNIX System Services Command Reference*.

5. Activate the automount facility from the shell. You can activate the automount facility in one of two ways. Base your choice on your particular situation.

If you want to start the automount facility . . .	Then . . .
From the shell Rule: You must have a superuser ID.	Issue: <code>/usr/sbin/automount</code> Rule: If the automount facility was started from the shell, do not submit any job that requires an automount-managed file system until automount initialization is complete. The entire syntax is as follows: <code>/usr/sbin/automount [-s] [Master filename]</code> If arguments are not used, the automount facility reads the /etc/auto.master and MapName files. If a master file name is specified, that file name is used instead of /etc/auto.master . The -s option only checks the syntax of the configuration file. The automount policy is not activated.
During an OMVS initialization	Add the following lines to the /etc/rc file: <code># Start the automount facility</code> <code>/usr/sbin/automount</code> Rule: If the automount has been started from /etc/rc , do not submit jobs that require an automount-managed file system before you get the message BPXI004I, or they may fail due to allocation errors.

When you are done, you have activated the automount facility from the shell.

What happens when you start the automount facility from the shell?

Figure 24 on page 152 shows what happens when the automount facility is started from the shell. It shows how file systems are automatically mounted. (**Tip:** To access the shell, type OMVS from ISPF option 6.)

```

# df
Mounted on      Filesystem      Avail/Total    Files    Status
/               (OMVS.ROOT)    1432/89280    0        Available
# /usr/sbin/automount
FOMF0107I Processing file /etc/u.map
FOMF0108I Managing directory /u
# df
Mounted on      Filesystem      Avail/Total    Files    Status
/u             (*AMD/u)       0/8           0        Available
/              (OMVS.ROOT)    1432/89280    0        Available
# cd /u/user1
# cd /u/slekka/testdir
# cd /u/rpetri
# df
Mounted on      Filesystem      Avail/Total    Files    Status
/u             (*AMD/u)       0/8           0        Available
/u/rpetri      (OMVS.RPETRI)  4256/4320     0        Available
/u/slekka      (OMVS.SLEKKA)  4232/4320     0        Available
/u/user1       (OMVS.USER1)   4232/4320     0        Available
/              (OMVS.ROOT)    1432/89280    0        Available
# ls -l /u
Total 496
drwxr-xr-x  2 RPETRI  OMVSGRP      0 Nov  2 09:59 rpetri
drwxr-xr-x  2 SLEKKA  OMVSGRP      0 Nov  1 09:47 slekka
drwx----- 2 ADMIN   OMVSGRP      0 Nov  7 09:07 user1
# chown user1 /u/user1
# ls -l /u
Total 496
drwxr-xr-x  2 RPETRI  OMVSGRP      0 Nov  2 09:59 rpetri
drwxr-xr-x  2 SLEKKA  OMVSGRP      0 Nov  1 09:47 slekka
drwx----- 2 USER1   OMVSGRP      0 Nov  7 09:07 user1
#

====>

ESC=¢ 1=Help  2=SubCmd 3=HlpRetrn 4=Top      5=Bottom 6=TS0
      7=BackScr 8=Scroll 9=NextSess 10=Refresh 11=FedRetr 12=Retrieve

```

Figure 24. Follow-up steps when using the automount facility

- 1** The **automount** command is being issued from a superuser ID to start the automount facility from the shell.
- 2** The automount facility scans the **/etc/auto.master** file first to see what MapName file or files should be read. Here, the **/u** directory is being managed. Calling the **automount** command twice by mistake does not cause problems regardless of whether a file system is already mounted. The automount facility reads the **/etc/auto.master** file and associated MapName file or files again and then picks up any changes.
- 3** The display free space command (**df**) is issued. It shows that the automount facility has been started and is managing the **/u** directory. Notice the **(*AMD/u)**.
- 4** Change directory (**cd**) commands are issued to access directories in the three file systems that are to be mounted from the **/u** directory. In this case, the directories **USER1**, **RPETRI**, and **SLEKKA** are used to resolve the **<uc_name>** symbol in the **/etc/u.map** file. The **RPETRI**, **SLEKKA**, and **USER1** directory names are translated to uppercase and substituted to build the HFS data set names **OMVS.RPETRI**, **OMVS.SLEKKA**, and **OMVS.USER1**, respectively. The **RPETRI**, **SLEKKA**, and **USER1** directories do not physically exist in any file system but will be created as pseudo mount points by the automount facility on which the HFS data sets **OMVS.RPETRI**, **OMVS.SLEKKA**, and **OMVS.USER1** are mounted.

5 Output from another **df** command shows that (*AMD/u) is managing the **/u** directory. It also shows that the OMVS.RPETRI, OMVS.SLEKKA and OMVS.USER1 data sets are now mounted at pseudo mount points **/u/rpetri**, **/u/slekka**, and **/u/user1**, respectively.

When automount is actively managing a particular mount point (in this case **/u**) you cannot add a file to this directory (**/u**) or create a new subdirectory off the **/u** directory using the **mkdir** command. If you try, you will see an allocation or catalog error.

6 The **ls -l /u** command is issued against the **/u** directory and the directory attributes are displayed.

7 The **chown** is issued to change the ownership of **/u/user1** directory from ADMIN to USER1.

8 The **ls -l /u** command is issued again to show that the owner field of the **/u/user1** directory is now set to USER1.

Figure 24 on page 152 shows how **<uc_name>** works with the **/etc/auto.master** and **/etc/u.map** files from Figure 22 on page 149. HFS data sets, OMVS.RPETRI, OMVS.SLEKKA and OMVS.USER1 have already been allocated. The low-level qualifier of the HFS data sets is the user ID which is also the directory mount point that **automount** will dynamically allocate. With the automount facility, if a user tries to access any directory in their HFS file system, the HFS data set is automatically mounted under the **/u** directory.

Tips:

- You can use specific entries for directory names when the parameters you want to use differ from the generic entry. Any parameters that are not specified are inherited from the generic entry. A specific entry defines a directory name called **wjs** in the name parameter of the MapName file rather than an ***** as shown in Figure 25 on page 154.

In this example, the duration for generic mounts is set to unmount idle file systems after 60 minutes. But in this specific mount entry, idle file systems will stay mounted indefinitely. Also, a specific file system is specified because the file system name does not conform to the format in the generic entry. All other attributes are inherited from the generic entry.

- To display the current automount policy, issue

```
/usr/sbin/automount -q
```
- You have two allocation options:
 - You can specify the allocation parameters when using **automount** to allocate an HFS data set. The automount map file specification has keywords that you can use to specify allocation keywords.
 - The mount points can be dynamically allocated by **automount**. When an HFS data set is first allocated for a new user, and the automount facility is used to dynamically allocate a mount point, the owner field is set to a superuser ID name.

Naming specific directories

Given the **/etc/auto.master** and **/etc/u.map** files as shown in Figure 25 on page 154, whenever the directory **/u/wjs** is referred to by a command such as **cd** or **cp**, the automount facility mounts HFS data set OMVS.WJS.HFS.

```

----- /etc/auto.master -----
/u          /etc/u.map

----- /etc/u.map -----
name        *
type        HFS
filesystem  OMVS.<uc_name>
mode        rdwr
duration    60
delay       10
/*
name        wjs
filesystem  OMVS.WJS.HFS
duration    nolimit

```

Figure 25. Specific entry in a MapName file

Restrictions: For generic and specific entries:

- Do not use a / in front of the name of the directory to be mounted in a specific entry in a MapName file. For example, in **/etc/u.map**, the following is correct:

```
name        wjs
```

But the following is not correct:

```
name        /wjs
```

- The directory name and the data set name qualifier for the HFS data set which is replaced by the variable **<uc_name>** have to be the same. Otherwise you will receive such error messages as:

```
EDC129I No such file or directory
or
EDC515I Dynamic allocation error
```

Changing which data sets get automounted

The automount facility is a physical file system (PFS) that is started with a FILESYSTYPE statement in the BPXPRMxx parmlib member. After the PFS is started, the automount facility manages the policy that you make active via the **/usr/sbin/automount** command. You can change the automount policy at any time, although you cannot set it to null. To change it, update the automount configuration files (**/etc/auto/master** and **MyMap**). Then activate that configuration with the **/usr/sbin/automount** command.

Stopping the automount facility

You cannot stop the automount facility after it has been started. Instead, change the automount configuration files (**/etc/auto/master** and **MyMap**) so that the automount facility is set to manage a dummy directory. Then activate that configuration with the **/usr/sbin/automount** command. Or you can simply unmount any *AMD/ file system.

Chapter 9. Shared HFS in a sysplex

Overview

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

In this chapter

This chapter covers the following subtasks.

Subtasks	Associated procedure (see . . .)
Establishing the root data set	"Creating the sysplex root HFS data set" on page 161
Establishing the HFS data set	"Creating the system-specific HFS data sets" on page 162
Mounting the version HFS	"Mounting the version HFS" on page 162
Creating the OMVS couple data set	"Creating an OMVS couple data set (CDS)" on page 164
Updating the COUPLExx data set	"Updating COUPLExx to define the OMVS CDS to XCF" on page 166
Keeping the automount policy consistent on all systems in the sysplex	"Keeping your automount policy consistent on all systems" on page 177

Although IBM recommends that you exploit shared HFS support, you are not required to. If you choose not to, you will continue to share HFS data sets as you have before OS/390 UNIX V2R9. To see how your file system structure differs in OS/390 UNIX V2R9 from V2R8, see "Comparing file systems in single system pre-OS/390 UNIX V2R9 and OS/390 UNIX V2R9 or later environments" on page 157.

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

What does shared HFS mean?

Sysplex users can access data throughout the file hierarchy.

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

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

With shared HFS, all file systems that are mounted by a system participating in shared HFS are available to all participating systems. In other words, once a file

system is mounted by a participating system, that file system is accessible by any other participating system. It is not possible to mount a file system so that it is restricted to just one of those systems. Consider a OS/390 UNIX V2R9 sysplex that consists of two systems, SY1 and SY2:

- A user logged onto any system can make changes to file systems mounted on **/u**, and those changes are visible to all systems.
- The system programmer who manages maintenance for the sysplex can change entries in both **/etc** file systems from either system.

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

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

How the end user views the HFS

This chapter describes the kinds of file systems and data sets that support the shared HFS capability in the sysplex. Figure 26 shows that, to the end users, the logical view of the HFS does not change for OS/390 UNIX V2R9. From their point of view, accessing files and directories in the system is just the same. That is true for all end users, whether they are in a sysplex or not.

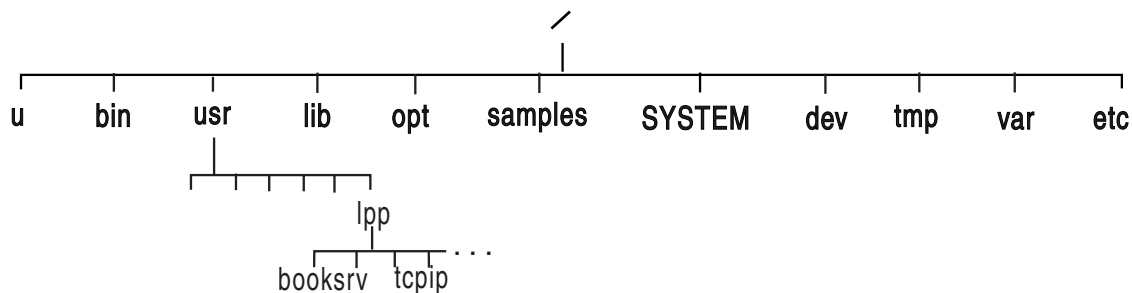


Figure 26. Logical view of shared HFS for the end user

This logical view applies to the end user only. However, system programmers need to know that the illustration of directories found in Figure 26 does not reflect the physical view of file systems. Starting in OS/390 UNIX V2R9, some of the directories are actually symbolic links, as is described in the following information.

Summary of new HFS data sets

This chapter introduces HFS data sets and terms needed to use shared HFS. Table 11 on page 157 summarizes the HFS data sets that are needed in a sysplex that has shared HFS. As you study the illustrations of file system configurations in this chapter, you can refer back to this table.

Table 11. HFS data sets that exist in a sysplex

Name	Characteristics	Purpose	How Created
Sysplex root	It contains directories and symbolic links that allow redirection of directories. Only one sysplex root HFS is allowed for all systems participating in shared HFS.	The sysplex root is used by the system to redirect addressing to other directories. It is very small and is mounted read-write. See "Procedures for establishing shared HFS in a sysplex" on page 161 for a more complete description of the sysplex root HFS.	The user runs the BPXISYSR job.
System specific	It contains data specific to each system, including the /dev , /tmp , /var , and /etc directories for one system. There is one system-specific HFS data set for each system participating in the shared HFS capability.	The system-specific HFS data set is used by the system to mount system-specific data. It contains the necessary mount points for system-specific data and the symbolic links to access sysplex-wide data, and should be mounted read-write. See "Creating the system-specific HFS data sets" on page 162 for a complete description of the system-specific HFS.	The user runs the BPXISYSS job on each participating system.
Version In a sysplex, <i>version HFS</i> is the new name for the root HFS.	It contains system code and binaries, including the /bin , /usr , /lib , /opt , and /samples directories. IBM delivers only one version root; you might define more as you add new system levels and new maintenance levels.	The version HFS has the same purpose as the root HFS in the non-sysplex world. It should be mounted read-only. See "Mounting the version HFS" on page 162 for a complete description of the version HFS.	IBM supplies this HFS in the ServerPac. CBPDO users create the HFS by following steps defined in the Program Directory.

Comparing file systems in single system pre-OS/390 UNIX V2R9 and OS/390 UNIX V2R9 or later environments

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

participating system or you might replace those individual file system BPXPRMxx members with a single file system BPXPRMxx member for all participating systems.

File systems in single system pre-OS/390 UNIX V2R9 environments

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

BPXPRMxx

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

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

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

Figure 27. BPXPRMxx for a single system before OS/390 UNIX V2R9 or later environments

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

Figure 28 shows the recommended setup of the root HFS in a single system environment.

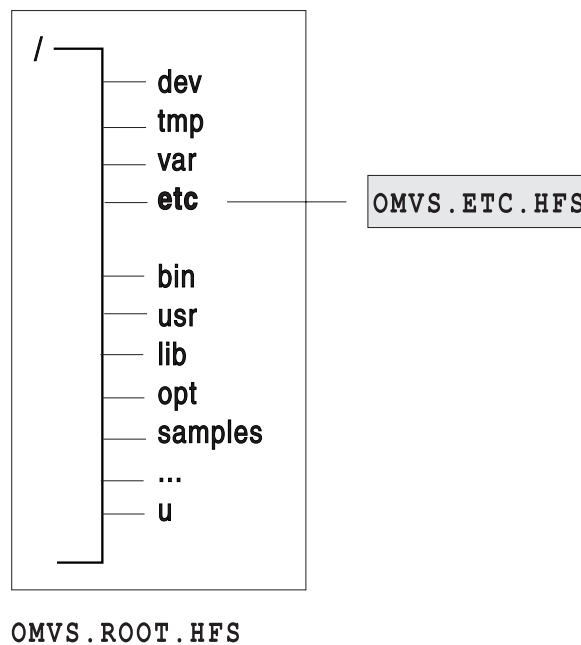


Figure 28. Single system before OS/390 UNIX V2R9

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

File systems in single system OS/390 UNIX V2R9 or later environments

Figure 29 shows what BPXPRMxx file system parameters would look like in an OS/390 UNIX V2R9 (or later) single system environment, and Figure 30 on page 160 shows the corresponding single system image. SYSPLEX(NO) is specified (or the default taken), and the mount mode is read-write.

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

BPXPRMxx

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

SYSPLEX(NO)

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

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

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

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

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

Figure 29. BPXPRMxx parmlib member for single system: OS/390 UNIX V2R9

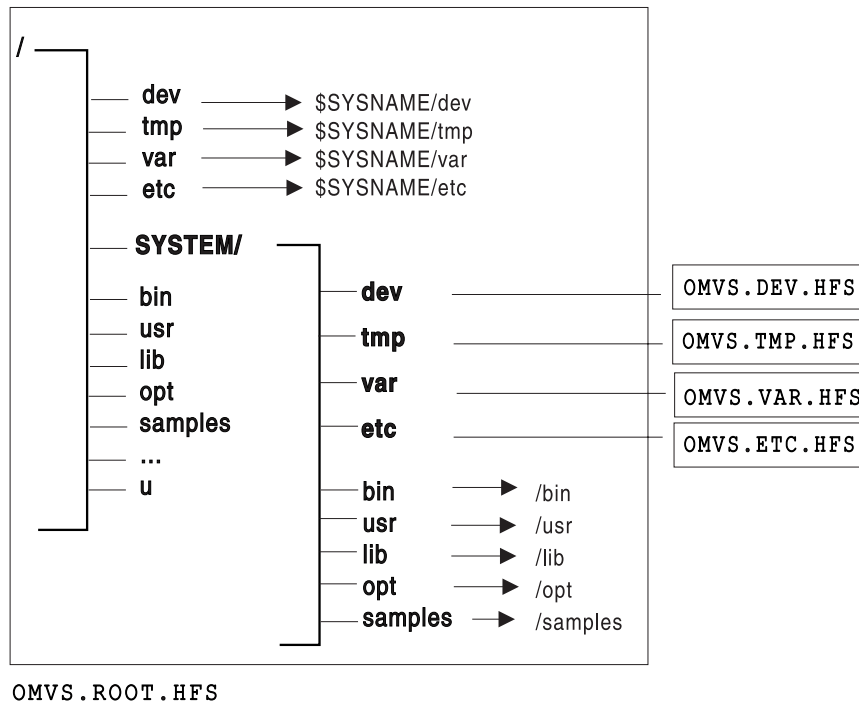


Figure 30. Single system: OS/390 UNIX V2R9

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

In Figure 30, the root file system contains an additional directory, `/SYSTEM`; existing directories, `/etc`, `/dev`, `/tmp` and `/var` are converted into symbolic links. These changes, however, are transparent to the user who brings up a single system environment.

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

File systems in OS/390 UNIX V2R9 or later sysplex environments

This section describes file systems in sysplex environments (OS/390 UNIX V2R9 or later) and what you need to do to take advantage of shared HFS support, such as creating specific HFS data sets (also see Table 11 on page 157) and the OMVS couple data set, updating `COUPLExx`, and customizing `BPXPRMxx`.

You must not assume that with shared HFS, two systems can share a common HFS data set for `/etc`, `/tmp`, `/var`, and `/dev`. This is not the case. Even with shared HFS, each system must have specific HFS data sets for each of these file systems. The file systems are then mounted under the system-specific HFS (see Figure 39 on page 173). With 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 SY2, you can gain read-write access to SY1's `/tmp` by specifying `/SY1/tmp/`.

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

BPXPRMxx member that contains file system information for each system participating in shared HFS. If you decide to define a BPXPRMxx member for each system, the FILESYSTYPE statements must be identical on each system. To see the differences between having one BPXPRMxx member for all participating systems and having one member for each participating system, see the two examples in “Scenario 2: Multiple systems in the sysplex using the same release level” on page 171.

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

Procedures for establishing shared HFS in a sysplex

Creating the sysplex root HFS data set

The sysplex root is an HFS data set that is used as the sysplex-wide root. This HFS data set must be mounted read-write and designated AUTOMOVE (see “Customizing BPXPRMxx for shared HFS” on page 166 for a description of the AUTOMOVE parameter in BPXPRMxx). Only one sysplex root is allowed for all systems participating in shared HFS. The sysplex root is created by invoking the BPXISYSR sample job in SYS1.SAMPLIB. After the job runs, the sysplex root file system structure would look like Figure 31:

Sysplex Root

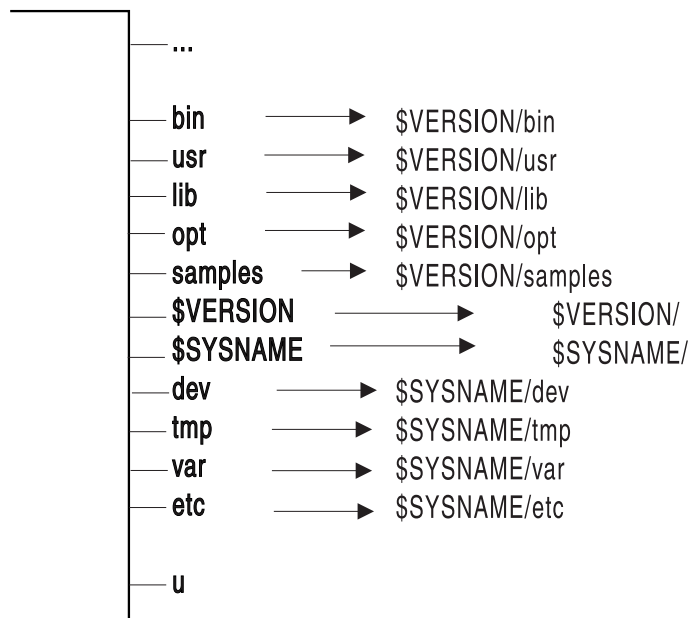


Figure 31. Sysplex root

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

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

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

Creating the system-specific HFS data sets

Directories in the system-specific HFS data set are used as mount points, specifically for **/etc**, **/var**, **/tmp**, and **/dev**. To create the system-specific HFS, run the BPXISYSS sample job in SYS1.SAMPLIB on each participating system (in other words, you must run the sample job separately for each system that will participate in shared HFS). After you invoke the job, the system-specific file system structure would look like Figure 32:

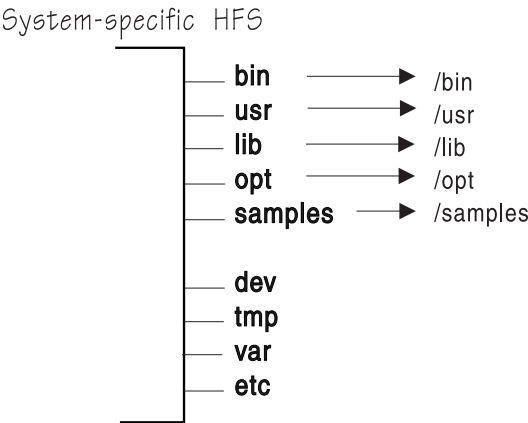


Figure 32. System HFS

The system-specific HFS data set should be mounted read-write, and should be designated NOAUTOMOVE. (See “Customizing BPXPRMxx for shared HFS” on page 166 for a description of the NOAUTOMOVE parameter in BPXPRMxx.)

Also, **/etc**, **/var**, **/tmp**, and **/dev** should be mounted NOAUTOMOVE. See “Shared HFS implications during a planned shutdown of z/OS UNIX” on page 180 for more information.

Recommendation: The name of the system-specific data set should contain the system name as one of the qualifiers. This enables you to use the &SYSNAME symbolic (defined in IEASYMxx) in BPXPRMxx.

If you mount a system-specific file system on other than the correct (system-specific) owner, either explicitly or due to AUTOMOVE=YES, loss of function may occur. For example, if the system-specific file system mounted at **/dev** for SY1 is moved to SY2 so that ownership is now SY2, the OMVS command on SY1 will fail.

Mounting the version HFS

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

Figure 33 on page 163 shows a version HFS.

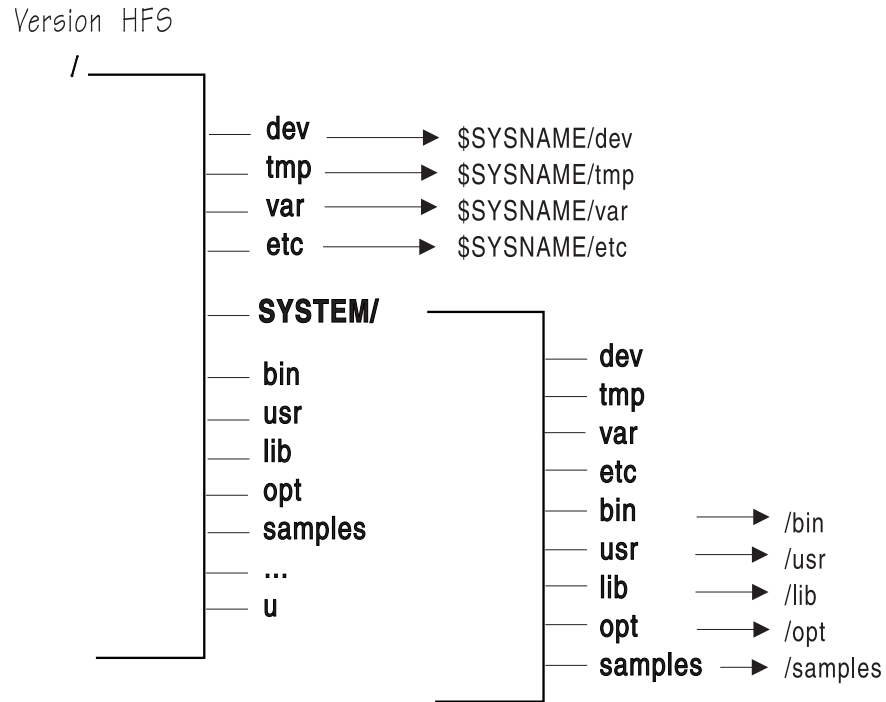


Figure 33. Version HFS

Recommendations:

1. Mount the version HFS read-only in a sysplex environment, and designate it AUTOMOVE. The mount point for the version HFS is dynamically created if the VERSION statement is used in BPXPRMxx.
2. Do not use &SYSNAME as one of the qualifiers for the version HFS data set name. In “Sysplex scenarios showing shared HFS capability” on page 168, REL9 and REL9A are used as qualifiers, which correspond to the system release levels. However, you do not necessarily have to use the same qualifiers. Other appropriate names are the name of the target zone, &SYSR1, or another qualifier meaningful to the system programmer.

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

Before you mount your version HFS read-only, you may have some element-specific actions. These are described in “Post-installation actions for mounting the root HFS in read-only mode” on page 121.

Using the automove system list (SYSLIST)

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 be moved to when the owning system leaves the sysplex. There are different ways to specify the automove system list.

- On the MOUNT statement in BPXPRMxx, specify the AUTOMOVE keyword, including the indicator and system list.
- For the TSO MOUNT command, specify the AUTOMOVE keyword, including the indicator and system list.
- Use the MOUNT shell command.
- Use the ISHELL MOUNT interface.
- Specify the MNTE_SYSLIST variable for REXX.
- Specify the indicator and system list for the automove option in the **chmount** shell command.
- Specify the indicator and system list for the automove option in the SETOMVS operator command.

Creating an OMVS couple data set (CDS)

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

Use of the CDS functions in the following manner:

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

This system processes its BPXPRMxx parmlib member, including all its ROOT and MOUNT statement information. It is also the designated owner of the byte range lock manager for the participating group. The MOUNT and ROOT information are logged in the CDS so that other systems that eventually join the participating group can read data about systems that are already using shared HFS.

2. Subsequent systems joining the participating group will read what is already logged in the CDS and will perform all mounts. Any new BPXPRMxx mounts are processed and logged into the CDS. Systems already in the participating group will then process the new mounts added to the CDS.

Following is the sample JCL with comments. The statements in bold contain the values that you specify based on your environment.

```
/*  
//STEP10 EXEC PGM=IXCLIDSU  
//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 OMVS.*/
   Default =    50
   Minimum =    50
   Maximum = 1000          */

ITEM NAME(DISTBRLM) NUMBER(1)
/*Enable conversion to a distributed BRLM.
   1, distributed BRLM enabled,
   0, distributed BRLM is not enabled during next sysplex IPL
   Default = 0  */
/* Begin definition for OMVS couple data set(2)          */
DEFINEDS SYSPLEX(PLEX1)
/* Name of the sysplex in which the OMVS couple data set is to be used.      */
DSN(SYS1.OMVS.CDS02) VOLSER(3390x2)
/* The name and volume for the OMVS couple data set. The utility will
   allocate a new data set by the namespecified on the volume specified.      */
MAXSYSTEMS(8)
/* Specifies the number of systems to be supported by the OMVS CDS.
   Default =      8          */
   NOCATALOG
/* Default is not to CATALOG */
DATA TYPE(BPXMCDs)
/* The type of data in the data set being created is for OMVS. BPXMCDs is the
   TYPE for OMVS.          */
ITEM NAME(MOUNTS) NUMBER(500)
/* Specifies the number of MOUNTS that can be supported by OMVS.
   Default =    100
   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          */
ITEM NAME(DISTBRLM) NUMBER(1)
/*Enables conversion to a distributed BRLM.
   1, distributed BRLM enabled,
   0, distributed BRLM is not enabled during next sysplex IPL
   Default = 0  */

```

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

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

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

Conversely, if the NUMBER values are too small, the function (for example, the number of mounts supported) would fail after the limit is reached. However, a new CDS can be formatted and switched in with larger values specified in NUMBER. To make the switch, issue the SETXCF COUPLE,PSWITCH command. For more information on this command, see the section on couple data set considerations in *z/OS MVS Setting Up a Sysplex*. The number of file systems required (factoring in an additional number to account for extra mounts), determines your minimum and maximum NUMBER value.

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

Updating COUPLExx to define the OMVS CDS to XCF

Update the active COUPLExx parmlib member to define a primary and alternate OMVS CDS to XCF. The primary and alternate CDSs should be placed on separate volumes. (The sample JCL in “Creating an OMVS couple data set (CDS)” on page 164 shows the primary CDS on volume 3390x1 and the secondary CDS on 3390x2.)

Figure 34 shows the COUPLExx parmlib member; statements that define the CDS are in bold.

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

Figure 34. COUPLExx parmlib member

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

Customizing BPXPRMxx for shared HFS

HFS sharing enables you to use one BPXPRMxx member to define all the file systems in the sysplex. This means that each participating system has its own BPXPRMxx member to define system limits, but shares a common BPXPRMxx member to define the file systems for the sysplex. This is done through the use of system symbolics. Figure 37 on page 171 shows an example of this unified

member. You can also have multiple BPXPRMxx members defining the file systems for individual systems in the sysplex. An example of this is Figure 38 on page 172.

The following parameters set up HFS sharing in a sysplex:

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

If **SYSPLEX(YES)** is specified in the BPXPRMxx member, but the system is initialized in XCF-local mode, either by specifying **COUPLE SYSPLEX(LOCAL)** in the COUPLExx member or by specifying **PLEXCFG=XCFLOCAL** in the IEASYSxx member, then the kernel will ignore the **SYSPLEX(YES)** value and initialize with **SYSPLEX(NO)**. This system will not participate in shared HFS support after the initialization completes.

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

There is one version HFS for every instance of the **VERSION** parameter. More information about version HFS appears in “Mounting the version HFS” on page 162.

- The **SYSNAME(sysname)** parameter on **ROOT** and **MOUNT** statements specifies the particular system on which a mount should be performed. *sysname* is a 1–8 alphanumeric name of the system. This system will then become the owner of the file system mounted. The owning system must be IPLed with **SYSPLEX(YES)**.

Recommendation: Specify **SYSNAME(&SYSNAME.)** or omit the **SYSNAME** parameter. In this case, the system that processes the mount request mounts the file system and becomes its owner.

The **SYSNAME** parameter is also used with **SETOMVS** when moving file systems, as demonstrated in “Moving file systems in a sysplex” on page 177.

- The **AUTOMOVE**, **NOAUTOMOVE**, and **UNMOUNT** parameters on **ROOT** and **MOUNT** indicate what happens to the file system if the system that owns that file system goes down.
 - **AUTOMOVE** specifies that ownership of the file system 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.

Tip: To ensure that the root is always available, use the default, which is **AUTOMOVE**.

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

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

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

Sysplex scenarios showing shared HFS capability

Scenario 1: First system in the sysplex

Figure 35 on page 169 and Figure 36 on page 170 shows a z/OS UNIX file system configuration for shared HFS. Here, SYSPLEX(YES) and a value on VERSION are specified, and a directory is dynamically created on which the version HFS data set is mounted. This type of configuration requires a sysplex root and system-specific HFS.

BPXPRMxx for (SY1)

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

VERSION('REL9')
SYSPLEX(YES)

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

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

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

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

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

Figure 35. BPXPRMxx parmlib setup — HFS sharing

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

2 This is the system-specific HFS data set, and was created by running the BPXISYSS job. It must be mounted read-write. NOAUTOMOVE is specified because this file system is system-specific and ownership of the file system should not move to another system should the owning system go down. The MOUNTPoint statement **/&SYSNAME.** will resolve to **/SY1** during parmlib processing. This mount point is created dynamically at system initialization.

3 This is the old root HFS (version HFS).

Recommendation: 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.

4 This HFS contains the system-specific **/dev** information. NOAUTOMOVE is specified because this file system is system-specific; ownership should not move

to another system should the owning system go down. The MOUNTPPOINT statement **/&SYSNAME./dev** will resolve to **/SY1/dev** during parmlib processing.

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

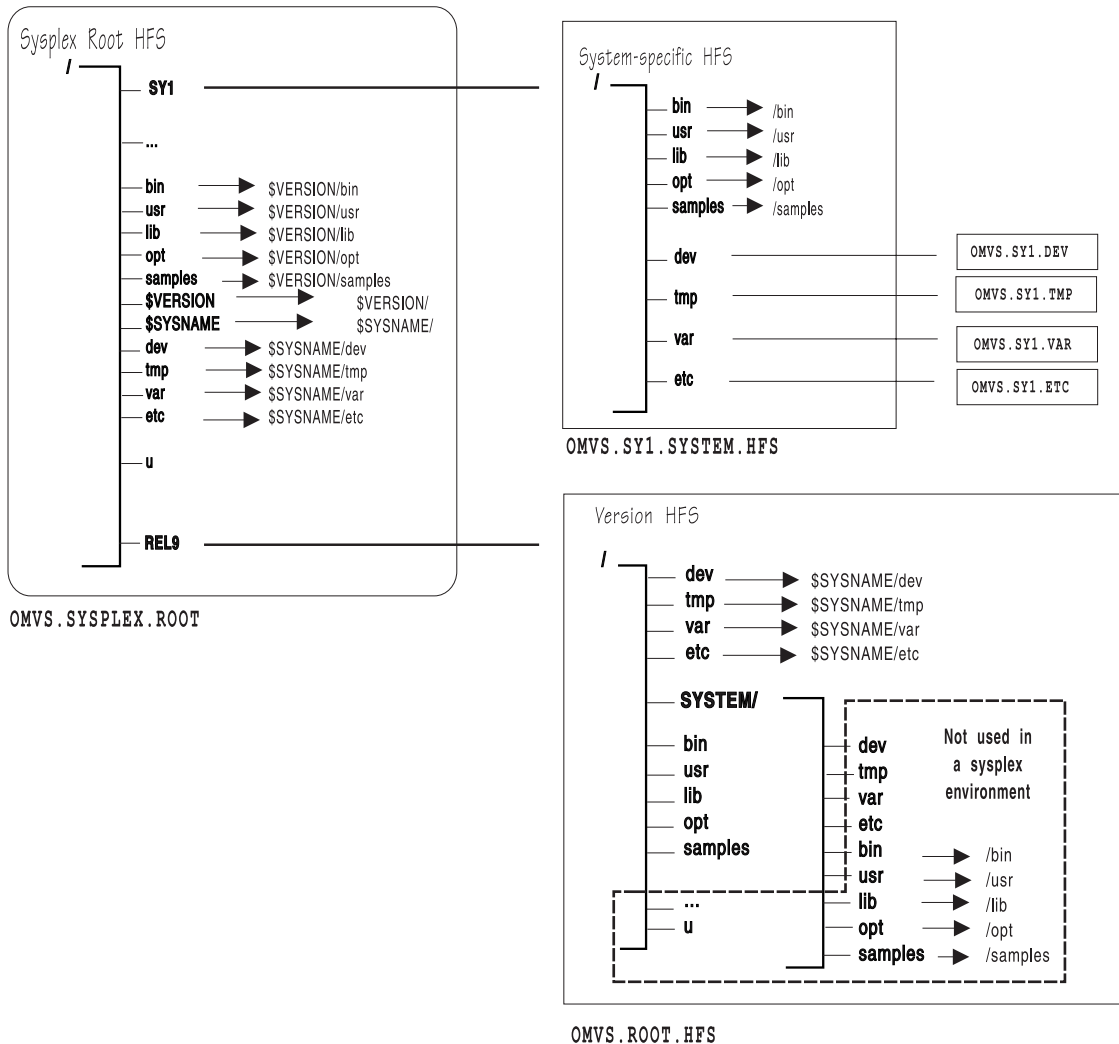


Figure 36. HFS sharing in a sysplex

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

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

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

Scenario 2: Multiple systems in the sysplex using the same release level

Figure 39 on page 173 shows another SYSPLEX(YES) configuration. In this configuration, however, two or more systems are sharing the same version HFS (the same release level of code). Figure 37 shows a sample BPXPRMxx for the entire sysplex (what IBM recommends) using `&SYSNAME`. as a symbolic name, and Figure 38 on page 172 shows a configuration where each system in the sysplex has its own BPXPRMxx. For our example, SY1 has its own BPXPRMxx and SY2 has its own BPXPRMxx.

One BPXPRMxx Member to Define File Systems for the Entire Sysplex

```

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

VERSION('REL9')
SYSPLEX(YES)

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

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

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

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

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

Figure 37. Sharing HFS data sets: one version HFS and one BPXPRMxx for the entire sysplex

BPXPRMS1 (for SY1)	BPXPRMS2 (for SY2)
FILESYSTYPE TYPE(HFS) ENTRYPOINT(GFUAINIT) PARM(' ')	FILESYSTYPE TYPE(HFS) ENTRYPOINT(GFUAINIT) PARM(' ')
VERSION('REL9') SYSPLEX(YES)	VERSION('REL9') SYSPLEX(YES)
ROOT FILESYSTEM('OMVS.SYSPLEX.ROOT') TYPE(HFS) MODE(RDWR)	ROOT FILESYSTEM('OMVS.SYSPLEX.ROOT') TYPE(HFS) MODE(RDWR)
MOUNT FILESYSTEM('OMVS.SY1.SYSTEM.HFS') TYPE(HFS) MODE(RDWR) NOAUTOMOVE MOUNTPOINT('/SY1')	MOUNT FILESYSTEM('OMVS.SY2.SYSTEM.HFS') TYPE(HFS) MODE(RDWR) NOAUTOMOVE MOUNTPOINT('/SY2')
MOUNT FILESYSTEM('OMVS.ROOT.HFS') TYPE(HFS) MODE(READ) MOUNTPOINT('/\$VERSION')	MOUNT FILESYSTEM('OMVS.ROOT.HFS') TYPE(HFS) MODE(READ) MOUNTPOINT('/\$VERSION')
MOUNT FILESYSTEM('OMVS.SY1.DEV') TYPE(HFS) MODE(RDWR) NOAUTOMOVE MOUNTPOINT('/SY1/dev')	MOUNT FILESYSTEM('OMVS.SY2.DEV') TYPE(HFS) MODE(RDWR) NOAUTOMOVE MOUNTPOINT('/SY2/dev')
MOUNT FILESYSTEM('OMVS.SY1.TMP') TYPE(HFS) MODE(RDWR) NOAUTOMOVE MOUNTPOINT('/SY1/tmp')	MOUNT FILESYSTEM('OMVS.SY2.TMP') TYPE(HFS) MODE(RDWR) NOAUTOMOVE MOUNTPOINT('/SY2/tmp')
.	
.	
.	

Figure 38. Sharing HFS data sets: one version HFS and separate BPXPRMxx members for each system in the sysplex

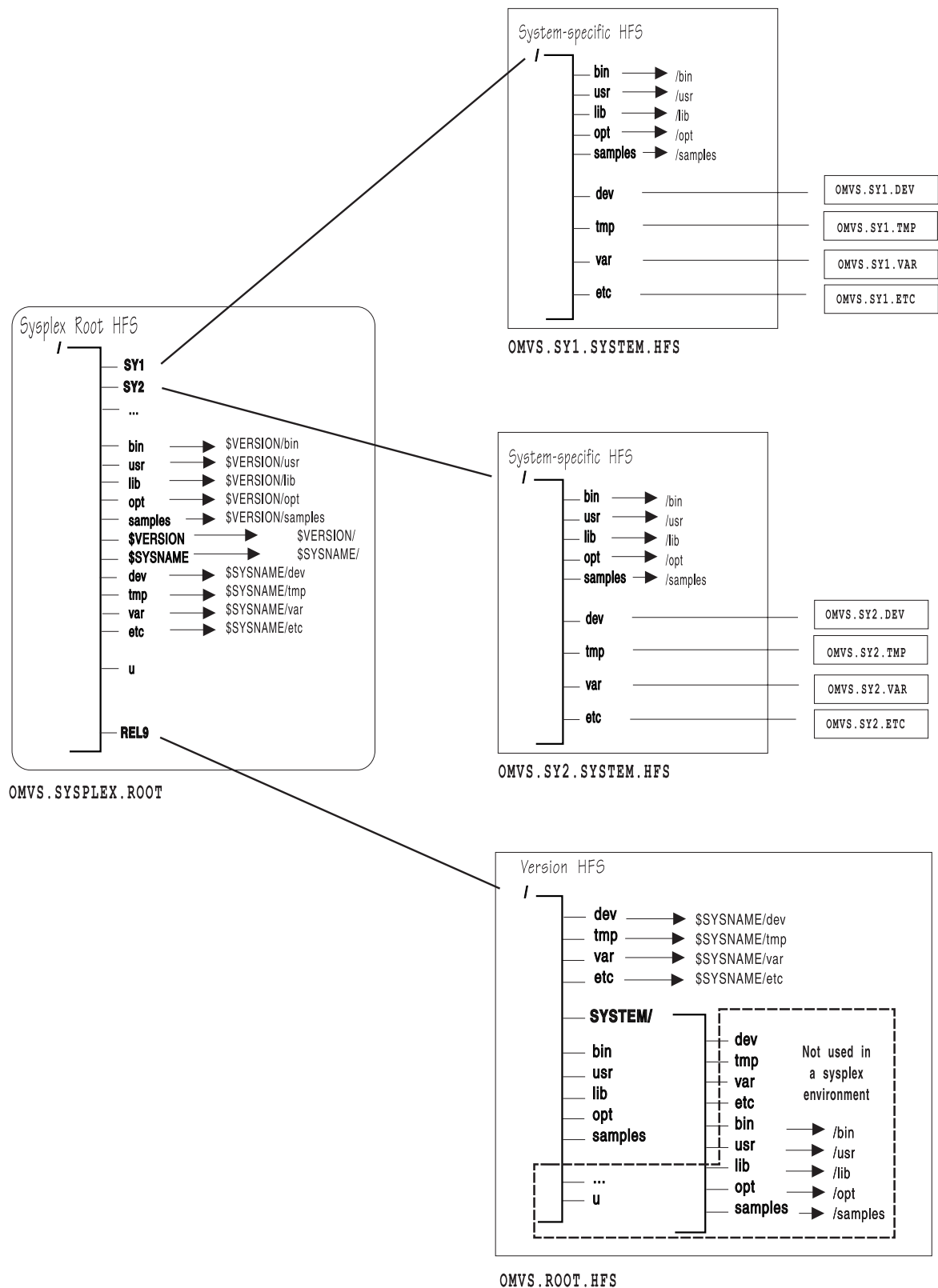


Figure 39. Sharing HFS data sets in a sysplex for OS/390 V2R9: multiple systems in a sysplex using the same release level

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

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

Scenario 3: Multiple systems in a sysplex using different release levels

If your participating group is in a sysplex that runs multiple levels of z/OS and/or OS/390, your configuration might look like the one in Figure 41 on page 175. In that configuration, each system is running a different level of z/OS and, therefore, has different version HFS data sets; SY1 has the version HFS named **OMVS.SYSR9A.ROOT.HFS** and SY2 has the version HFS named **OMVS.SYSR9.ROOT.HFS**. Figure 40 shows two **BPXPRMxx** parmlib members that define the file systems in this configuration. Figure 42 on page 176 shows a single **BPXPRMxx** parmlib member that can be used to define this same configuration; it uses **&SYSR1.** as the symbolic name for the two version HFS data sets.

BPXPRMxx (for SY1)	BPXPRMxx (for SY2)
FILESYSTYPE TYPE(HFS) ENTRYPOINT(GFUAINIT) PARM(' ')	FILESYSTYPE TYPE(HFS) ENTRYPOINT(GFUAINIT) PARM(' ')
VERSION('REL9A') SYSPLEX(YES)	VERSION('REL9') SYSPLEX(YES)
ROOT FILESYSTEM('OMVS.SYSPLEX.ROOT') TYPE(HFS) MODE(RDWR)	ROOT FILESYSTEM('OMVS.SYSPLEX.ROOT') TYPE(HFS) MODE(RDWR)
MOUNT FILESYSTEM('OMVS.&SYSNAME..SYSTEM.HFS') TYPE(HFS) MODE(RDWR) NOAUTOMOVE MOUNTPPOINT('/&SYSNAME.')	MOUNT FILESYSTEM('OMVS.&SYSNAME..SYSTEM.HFS') TYPE(HFS) MODE(RDWR) NOAUTOMOVE MOUNTPPOINT('/&SYSNAME.')
MOUNT FILESYSTEM('OMVS.SYSR9A.ROOT.HFS') TYPE(HFS) MODE(READ) MOUNTPPOINT('/\$VERSION')	MOUNT FILESYSTEM('OMVS.SYSR9.ROOT.HFS') TYPE(HFS) MODE(READ) MOUNTPPOINT('/\$VERSION')
MOUNT FILESYSTEM('OMVS.&SYSNAME..DEV') TYPE(HFS) MODE(RDWR) NOAUTOMOVE MOUNTPPOINT('/&SYSNAME./dev')	MOUNT FILESYSTEM('OMVS.&SYSNAME..DEV') TYPE(HFS) MODE(RDWR) NOAUTOMOVE MOUNTPPOINT('/&SYSNAME./dev')
MOUNT FILESYSTEM('OMVS.&SYSNAME..TMP') TYPE(HFS) MODE(RDWR) NOAUTOMOVE MOUNTPPOINT('/&SYSNAME./tmp')	MOUNT FILESYSTEM('OMVS.&SYSNAME..TMP') TYPE(HFS) MODE(RDWR) NOAUTOMOVE MOUNTPPOINT('/&SYSNAME./tmp')
.	
.	
.	

Figure 40. *BPXPRMxx* parmlib setup for multiple systems sharing HFS data sets and using different release levels

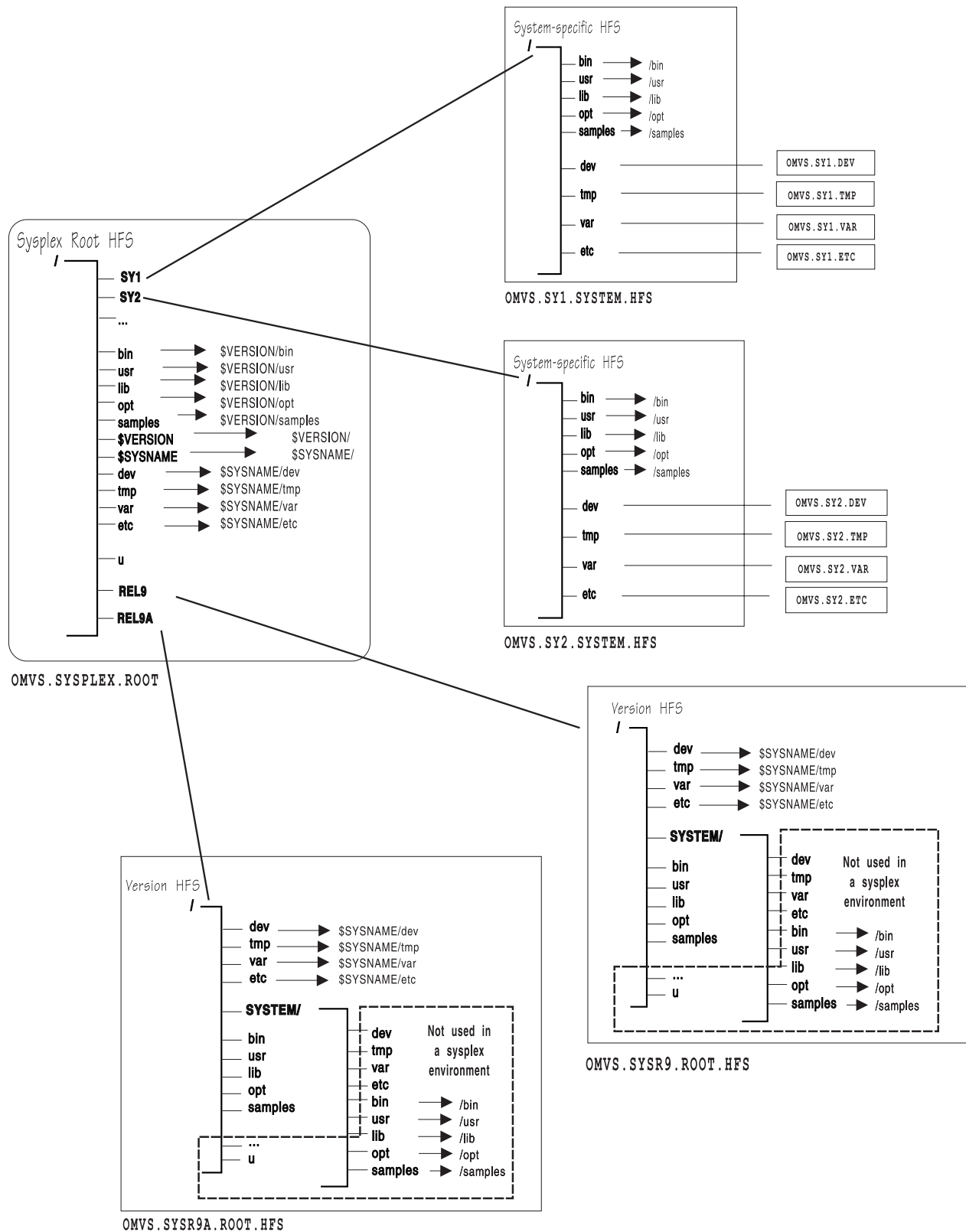


Figure 41. Sharing HFS data sets between multiple systems using different release levels

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

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

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

**One BPXPRMxx Member to define file systems for the entire sysplex
Using different releases**

```

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

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

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

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

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

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

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

```

Figure 42. One BPXPRMxx parmlib member for multiple systems sharing HFS data sets and using different release levels

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

Keeping automount policies consistent on all systems in the sysplex

Rule: You must keep the automount policies consistent across all the participating systems in the sysplex. The automount facility will not manage any directory until it can process the entire policy without encountering any errors.

Keeping your automount policy consistent on all systems

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

For example both SY1 and SY2 have the following files:

- **/etc/auto.master**
 /u /etc/u.map
- **/etc/u.map**
 name *
 type HFS
 filesystem OMVS.<uc_name>.HFS
 mode rdwr
 duration 60
 delay 60

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

Moving file systems in a sysplex

You may need to change ownership of the file system for recovery or re-IPLing.

Tips:

- To check for file systems that have already been mounted, use the **df** command from the shell.
- The SETOMVS command used with the FILESYS, FILESYSTEM, mount point and SYSNAME parameters can be used to move a file system in a sysplex, or you can use the **chmount** command from the shell. However, do not move two types of file systems:
 - System-specific file systems
 - File systems that are being exported by DFS. You have to unexport them from DFS first and then move them

Examples:

1. To move ownership of the file system that contains **/u/wjs** to SY1:
 chmount -d SY1 /u/wjs
2. To move ownership of the payroll file system from the current owner to SY2 using SETOMVS, issue:
 SETOMVS FILESYS,FILESYSTEM='POSIX.PAYROLL.HFS',SYSNAME=SY2

 or (assuming the mount point is over directory **/PAYROLL**)
 SETOMVS FILESYS,mountpoint='/PAYROLL',SYSNAME=SY2

If you mount a system-specific file system on other than the correct (system-specific) owner, either explicitly or due to AUTOMOVE=YES, loss of function may occur. For example, if the system-specific file system mounted at /dev for SY1 is moved to SY2 so that ownership is now SY2, the OMVS command on SY1 will fail.

Also, opened FIFO files are automatically closed before the file system containing the FIFO is moved. They are closed because the in-storage FIFO data on the old system is not moved and is no longer accessible on new owning system.

Shared HFS implications during system failures and recovery

File system recovery in a shared HFS environment takes into consideration file system specifications such as AUTOMOVE, NOAUTOMOVE, 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. (The BPX1CLO and BPX1OPN callable services are discussed in *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.)

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

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

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

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

Do not mount AUTOMOVE file systems within NOAUTOMOVE file systems. When a NOAUTOMOVE file system becomes unowned and there are AUTOMOVE file systems mounted within it, those AUTOMOVE file systems will retain a level of availability, but only for files that are already open. When the NOAUTOMOVE file

system becomes unowned, it will not be possible to perform path name lookup through it to the file systems mounted within it, which will make those file systems unavailable for new access. When ownership is restored to the unowned file system, access to the file systems mounted within it is also restored.

Managing the movement of data

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

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

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

Table 12. Automove options supported by the MOUNT command

UNMOUNT	Attempts will not be made to keep the file system active when the current owner fails. The file system will be unmounted when the owner is no longer active in the participating group, as well as all the file systems mounted within it. The UNMOUNT option is available on z/OS V1R3 and higher. It is recommended for use on parmlib mounts for system-specific file systems, such as those that would be mounted at /etc , /dev , /tmp and /var .
NOAUTOMOVE	Attempts will not be made to keep the file system active when the current owner fails. The file system will remain in the hierarchy for possible recovery when the original owner reinitializes. Use this option on mounts for system-specific file systems if you want to have automatic recovery when the original owner rejoins the participating group. When the NOAUTOMOVE option is used, the file system becomes unowned when the owning system exits the participating group. The file system remains unowned until the last owning system restarts, or until the file system is unmounted. Because the file system still exists in the file system hierarchy, the file system mount point is still in use. An unowned file system is a mounted file system that does not have an owner. Because it still exists in the file system hierarchy, it can be recovered or unmounted.
AUTOMOVE	Recovery of the file system is to be performed to a randomly selected owner when the current owner fails. This option is recommended for use on mounts of file systems that are critical to operation across all the systems in the participating group. AUTOMOVE is the default.

Table 12. Automove options supported by the MOUNT command (continued)

AUTOMOVE with the exclude or include option	AUTOMOVE(EXCLUDE INCLUDE,sysname1,sysname2,...,sysnameN) specifies managed recovery of the file system if the current owner fails. - Use the EXCLUDE list to prevent recovery of a file system from transferring ownership to a particular system, or set of systems, in the participating group. When the current owner fails, recovery of the file system is performed to a randomly selected owner outside the exclude list. - Use the INCLUDE list to ensure that recovery of a file system will transfer ownership only to a particular system or set of systems in the participating group. Recovery of the file system is performed in priority order only by the list of systems specified in the INCLUDE list. Restriction: Do not use this option in a shared HFS environment with releases running below the V1R4 level of service. Only use this option on mounts of file systems that are critical to operation across a subset of systems in the participating group, or when you do not want certain systems in the participating group to have ownership of the file system. If recovery processing fails to establish a new owner for the file system, the file system is unmounted, along with all the file systems mounted within it.

Most of the z/OS UNIX interfaces that provide for mounting file systems (such as TSO, shell, ISHELL, and BPX2MNT) support some form of the options described in Table 12 on page 179. See the associated documentation for the exact syntax.

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

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

Shared HFS implications during a planned shutdown of z/OS UNIX

“Steps for shutting down z/OS UNIX using F OMVS,SHUTDOWN” on page 257 and “Steps for shutting down z/OS UNIX using F BPXOINIT,SHUTDOWN=...” on page 252 contain the procedures to use when you are shutting down z/OS UNIX. It is important that you understand the system actions that result when you use those procedures.

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

- A system failure does not provide any means for manual intervention. The automove option provides a set of rules for automatic recovery.

- A request to move a file system manually is a deliberate action on behalf of an authorized user or administrator, and should override any rules for automatic recovery.
- Using tools to prepare for a system outage is also a deliberate action on behalf of an authorized user or administrator, but you are using these tools in an environment that can be customized to allow for additional manual intervention. You can synchronize data before the system outage, and then manage the planned outage in the same way as the unplanned outage, by making use of the automatic recovery rules that are supplied by the automove options. If you prefer some other action, you can perform manual intervention to move specific file system ownership before you use these methods for shutdown preparation.

Use F OMVS,SHUTDOWN to shut down file systems. It is described in “What F OMVS,SHUTDOWN does” on page 256. If this is not appropriate for your installation, use the F BPXOINIT,SHUTDOWN=... procedure described in “Planned shutdowns using F BPXOINIT,SHUTDOWN=...” on page 252.

State of file systems after shutdown

File systems on the system where the shutdown was issued are immediately unmounted. As a result, data is synched to disk. For shared 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.

File system initialization

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

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

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

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

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

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

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

Locking files in the sysplex

You can lock all or part of a file that you are accessing for read-write purposes by using the byte range lock manager (BRLM).

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

When a system failure occurs on the system owning the BRLM, all history of byte range locks is lost. A new BRLM is established by one of the surviving systems in the sysplex, and locking can begin once that recovery has completed. However, to maintain locking integrity for files open on surviving systems after the system that owns the BRLM goes down, z/OS UNIX prevents further locking or I/O on opened files that were locked. In addition, the applications are signalled, just in case they never issue locking requests or I/O. Running applications that did not issue locking requests and did not have files open are not affected. After a failure where byte range locks are lost, z/OS UNIX provides the following information to processes that have used byte range locking:

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

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

Using distributed BRLM

Centralized BRLM is set up as the default. You can choose to have distributed BRLM initialized on every system in the sysplex. Each BRLM is responsible for handling locking requests for files whose file systems are mounted locally in that system. If you want distributed BRLM, you need to follow the steps in “Steps for setting up distributed BRLM on every system in the sysplex.”

Recommendation: Use distributed BRLM if you have applications that lock files that are mounted and owned locally. Examples are the **inetd** and **cron** daemons. Distributed BRLM eliminates a single point of failure by having applications like **inetd**, **cron**, and Domino server send their lock requests to the local distributed BRLM server instead of the remote central BRLM server.

Restrictions: After you set up distributed BRLM, the following restrictions apply:

- You cannot go back to centralized BRLM unless you restore the sysplex to the state it was in before distributed BRLM was set up. This will require resetting the couple data set, followed by a sysplex-wide IPL
- If any open file in the file system has been locked by BRLM, you cannot move file systems by an external command such as the following:

```
SETOMVS FILESYS, FILESYSTEM=,SYSNAME=
```

Before the move can succeed, the file must be closed.

- With distributed BRLM, certain cross-system deadlock scenarios may not be detected. Locking applications must ensure that they do not cause deadlocks.

Steps for setting up distributed BRLM on every system in the sysplex

Before you begin: You must bring down any locking applications if you are activating distributed BRLM in a running sysplex. If you do not, they will be exposed to any locking and I/O errors for any files that they have open.

Perform the following steps to set up BRLM on every system in the sysplex.

1. Ensure that all systems in the sysplex support BRLM.

```
F BPX0INIT,FILESYS=DISPLAY,GLOBAL
```

In the resulting display, you can check the supported levels. You should see v.p.m, with either m>5 or p>2, or v>1. If a system is not at these levels, you will have to apply APAR OW52293, with these corresponding PTFs: R609 UW85157, R703 UW85155, and R705 UW85156. (R706 automatically includes support equivalent to those PTFs.)

-
2. Update the BPXMCDs couple data set using the IXCL1DSU utility.

Tip: See “Creating an OMVS couple data set (CDS)” on page 164 for an example of the COUPLExx parmlib member. *z/OS MVS Setting Up a Sysplex* discusses the BPXMCDs couple data set and the IXCL1DSU utility.

-
3. Switch the CDS into a running sysplex.

Example:

```
SETXCF COUPLE,PSWITCH,TYPE=BPXMCDs
```

4. Remove the central BRLM server from the sysplex. Bringing it down eliminates all file locking history in the sysplex and allows the new distributed BRLM servers to start with a clean locking history.

When you are done, you have set up distributed BRLM. The system owning the file system will be the system that receives all locking requests for files in that file system.

Mounting file systems using symbolic links

You can mount different file systems at a logical mount point that resolves to a different path name on different systems.

While `$VERSION/` can be used to differentiate a path based on the version level of a system and `$SYSNAME/` can be used to differentiate on each system, you can use special identifiers to mount file systems using symbolic links. These are `$SYSSYMR/template` and `$SYSSYMA/template`.

Restrictions:

1. Like `$VERSION/` and `$SYSNAME/`, the identifiers need to be at the beginning of the link name.
2. Only the first occurrence of `$SYSSYMR/` or `$SYSSYMA/` in the link name will be recognized as an identifier for which the remaining text requires substitutions. Any other identifiers after the first one will remain as is in the resolved linkname.
3. Text must follow a `$SYSSYMR/` or `$SYSSYMA/` in order for it to be recognized as a valid identifier with text containing symbols to be resolved.
4. Any system symbol in the symbolic link text that is recognized by the ASASYMBM service will be resolved. However, only static system symbols should be used in order to avoid unexpected results. These symbols are assigned a value at initialization. For information about system symbols, see *z/OS MVS Initialization and Tuning Reference*.

Tip: You can use the `D SYMBOLS` command to display the current settings of system symbols.

Examples

These examples assume that the standard MVS symbol `&YSR1.` resolves to `OSV315` on `SY1` and resolves to `OSV315B` on `SY2`.

1. If the symbolic link is `/x/y/sym1`, and the symbolic link contains `$SYSSYMR/&YSR1./resdir`, a path name lookup on `/x/y/sym1` from `SY1` will resolve the symbolic link to `OSV315/resdir`. Because it is a relative path name (the identifier was `$SYSSYMR/`), the resulting path name will be `/x/y/OSV315/resdir`.

Example: On a mount, passing `/x/y/sym1` as the input mount point path name, the mount point would be: `/x/y/OSV315/resdir` on `SY1`.

- If the symbol `&YSR1.` resolves to `OSV315B` on `SY2`, a lookup of the same path name would result in a mount point of `/x/y/OSV315B/resdir`.
- On a `v_readlink` syscall, passing the `VnToken` for the symbolic link, the output linkname would be `OSV315/resdir` on `SY1` or `OSV315B/resdir` on `SY2`.

2. If the symbolic link is `/x/y/sym1`, and the symbolic link contains `$SYSSYMA/&YSR1./resdir`, a path name lookup on `/x/y/sym1` from `SY1` will

resolve the symbolic link to /OSV315/resdir. Because it is an absolute path name (the identifier was \$SYSSYMA/), the resulting path name will be /OSV315/resdir.

Example: On a mount, passing /x/y/sym1 as the input mount point path name, the mount point would be /OSV315/resdir on SY1.

- If the symbol &SYSR1. resolves to OSV315B on SY2, a lookup of the same path name from SY2 would result in a mount point of /OSV315B/resdir.
- On a v_readlink syscall, passing the VnToken for the symbolic link, the output linkname would be /OSV315/resdir on SY1 and /OSV315B/resdir on SY2.

Mounting file systems using NFS client mounts

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

In a similar way, the z/OS NFS client gives users remote access to files on an NFS server. Using NFS, the user can mount all or part of the remote file system and make it appear as part of the local z/OS file hierarchy. From there, the user can create, delete, read, write, and treat the remotely located files as part of their own file system.

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

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

File system availability

In the shared HFS environment, file system availability and accessibility depend on a number of important factors. These factors can vary depending on how a file system is mounted and the capability of the file system to manage itself in a sysplex environment. After you set up the shared HFS environment for cross-system communication ("Procedures for establishing shared HFS in a sysplex" on page 161), it will be helpful to understand how file systems availability is provided to your systems, and what kinds of actions can cause interruptions to that availability.

Minimum setup required for file system availability

Rules:

- For DASD file systems, at least one system in the shared HFS group needs to have a physical I/O path to the volume where the file system resides and the volume is varied online. Without connectivity from at least one system, the file system will not be available to any of the systems in the shared HFS group. Connectivity from one system can provide shared HFS accessibility to the file system for all other systems in the shared HFS group.
- All systems need to have the physical file system (PFS) started. Accomplish this by placing the appropriate FILESTYPE statement in the BPXPRMxx parmlib

member that is used in the configuration. Additionally, any necessary subsystems required by the PFS must be started and configured, especially if this system is to function as the file system owner. For example, the NFS Client PFS requires that the TCP/IP subsystem be started and a network connection configured.

Read-write connections for non-sysplex aware file systems

Most physical file systems (PFSes) allow only one connection for update at a time. Such file systems are called *non-sysplex aware for update*. This is directly related to the mount mode of the file system. With HFS, for example, only one system can actually connect to the file system with a mode of RDWR. That system is called the *file system owner*.

The other systems that want to participate in shared HFS sharing for the HFS file systems will also request a RDWR mount, but their access will be provided via cross-system messaging with the file system owner which has already established the read-write connection. These systems are called *file system clients*. When the file system owner becomes unavailable (for example, through system shutdown), it will be important for another system (one of the file system clients) to have the file system volume varied online so that a new owner can be established. This helps ensure maximum file system availability in the shared HFS group.

Read-write connections for sysplex-aware file systems

Some PFSes can handle multiple concurrent connections for update. They are capable of managing the serialization of such requests. Such file systems are called *sysplex aware for update*. Most network file systems have this capability. NFS Client is one such file system type.

For a read-write mount to NFS Client, each system in the shared HFS group will make a direct connection to NFS. The first system to make such a connection is still called the file system owner. All subsequent systems to make a direct connection are considered non-owners, rather than clients. This type of multiple direct connection for read-write access allows for maximum I/O performance by eliminating the need to send requests to the file system owner.

However, sometimes a non-owning system cannot make a direct connection to the PFS even after meeting the minimum requirements (for example, sometimes requests to NFS Client time out before they are satisfied). That system might be given a cross-system messaging connection, making it a client to the file system. While this is not the optimal mount mode for this type of file system, it does allow access to the file system.

Read-only connections for non-sysplex aware file systems

There may be some PFSes that do not support multiple concurrent connections for read-only access. These are called *non-sysplex aware for readonly*, and are handled the same as the read-write connections for non-sysplex aware file systems.

Read-only connections for sysplex-aware file systems

PFSes that support multiple concurrent connections for read-only access are called *sysplex aware for readonly*. The HFS PFS falls into this category. Such file systems are handled the same as the read-write connections for sysplex aware file systems. The read-only connections are attempted locally for each system in the shared HFS group, but if the file system volume is not online to a system, then the system becomes a client to the file system via cross-system messaging with the owner.

Situations that can interrupt availability

Some situations may cause interruptions to file system availability on one or more systems. Following is a list of some of the most common causes. It is not meant to be an exhaustive list.

- **Loss of the file system owner.** If the file system owner leaves the shared HFS group (through system failure, soft shutdown, VARY, XCF, OFFLINE, or some other means), an attempt may be made to establish another file system owner if requested by the AUTOMOVE specification of the mount. If a new file system owner cannot be established, the file system will become unowned. It will be unavailable until the original owner can reclaim it, or until another owner is established through subsequent automated recovery actions performed by shared HFS.
- **PFS termination.** If a PFS terminates on one system, it can affect file system availability on other systems.
 - Prior to V1R2, if a PFS terminates on one system, all file systems of that type are unmounted across the sysplex.
 - In V1R2 and later, if a PFS terminates on one system, any file systems of that type that are owned by other systems are not affected. File systems of that type are moved to new owners whenever possible if they are owned by the system where the PFS is terminating and are automovable. These file systems remain accessible to other systems. If they cannot be moved to new owners, they are unmounted across the sysplex. It may not be possible to move a file system due to a lack of connectivity from other systems, or if the file system containing the mount point for the file system needed to be moved but could not be.
- **VARY volume,OFFLINE.** When the volume for a file system is varied offline, it will make that file system inaccessible to that system. However, if the volume is online to other systems, it may still be accessible to those systems and to other systems via cross-system messaging. This would be the case for sysplex-aware file systems for read-write or read-only access. Unlike loss of the file system owner, varying a file system volume offline will not result in any attempt by the system to restore accessibility to systems on which it is lost.

Tuning z/OS UNIX performance in a sysplex

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

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

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

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

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

Thus, adding z/OS UNIX to a sysplex increases XCF message traffic. To control this traffic, closely monitor the number and size of message buffers and the number of message paths within the sysplex. It is likely that you will need to define additional

XCF paths and increase the number of XCF message buffers above the minimum default. For more information on signaling services in a sysplex environment, see *z/OS MVS Setting Up a Sysplex*.

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

DFS considerations

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

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

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

Chapter 10. Customizing the shells and utilities

Overview

This chapter discusses how to customize the z/OS and tcsh shells and the common tasks that need to be done when setting up the utilities.

Connecting to the shell

To work interactively, the shell user connects to the system in one of the following ways:

- Logs on to TSO/E and enters the TSO/E command OMVS, which invokes a shell. The OMVS command provides a 3270 terminal interface to the shell, and you can use the options to customize the interface—for example, function key settings.
- Issues the **rlogin** command, which invokes the shell. It provides an asynchronous terminal interface to the shell, familiar to UNIX users.
- Issues the **telnet** command, which invokes the shell. It provides an asynchronous terminal interface to the shell, which is familiar to UNIX users.

See *z/OS UNIX System Services User's Guide* for a description of these interfaces to the shell.

After the user logs in to the shell, the system initializes the shell. See “Customizing the z/OS UNIX shells” on page 191 for information about customizing the shell invocation.

Invoking the shell automatically under TSO/E

A shell user can invoke the shell automatically at the end of logging on to TSO/E. For the automatic invocations, the system invokes the shell after the TSO/E logon completes initialization. Automatic invocation is a handy way to enter the same OMVS command with options each time.

The automatic invocation can be set up by the system programmer or by the shell user.

By system programmer action

For a system programmer to enable shell users to invoke the shell automatically when they log on to TSO/E, do the following:

1. Select or create a TSO/E logon procedure for users who wish to invoke the shell automatically when they log on to TSO/E.
2. In the logon procedure, add a PARM parameter to the EXEC statement for program IKJEFT01, as follows:

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

3. Tell users to specify the procedure on the TSO/E logon panel on the line:

Procedure ===>

To customize the OMVS command for all shell users, you can create a REXX exec with the customized options. Then specify the name of the REXX exec in the PARM parameter, instead of with the OMVS command. In the exec, for example, you can specify changes like these:

- Use of the 3270 alarm.

- Number of sessions. (The default is 1.) For example:
OMVS SESSIONS (3)
- The key or keys to be used for escape.
- The settings for the function keys.
- The table to be used for code page conversion.
- Shared address space.

To customize any of the default function key settings, type your selection within the parentheses after the function key name. For example, to make function key 1 (<PF1>) the Control key, issue:

```
OMVS PF1(CONTROL)
```

You use it to type an escape sequence such as <Ctrl-D>. (First you type d on the command line, and then you press the function key.)

By TSO/E user action

A TSO/E user can invoke the shell automatically when logging on to TSO/E. When logging on, you can invoke the shell by adding one of the following to the TSO/O logon panel:

- The OMVS command. For example, if the default options are desired:

```
Command ==> OMVS
```

Or:

- The name of the REXX exec that contains an OMVS command with operands for desired options. For example, if the exec name is MYOM:

```
Command ==> MYOM
```

TSO/E processes this command each time the user logs on until the user deletes the command from the panel or changes it.

You may want to invoke OMVS from ISPF for the following reasons:

- It is faster to invoke an ISPF dialog such as OEDIT or OBROWSE because ISPF does not need to start and stop to run the command.
- It is not necessary to type *** and press <Enter> after running an ISPF dialog; control returns to the shell.
- You can use split-screen support when using an ISPF dialog.

See *z/OS TSO/E Administration* for information about the REXX exec.

Determining the CPU time limit

The time limit for using a shell is the same as the TSO/E timeout. In determining the time, the system does not count the processing time for shells running in separate address spaces or processes forked by the shell. If you specify environment variable `_BPX_SHAREAS=YES`, then the shell processes and possibly one shell command are created in local processes. The CPU time consumed by local processes comes out of the TSO/E address space's time limit.

Supplying an alternative shell

If your installation decides to supply its own shell, consider doing the following:

- Install the alternative shell in the file system.
- Set the sticky bit on and put the alternative shell in the link pack area (LPA).

- Specify the pathname of the shell in the **PROGRAM** parameter for the OMVS RACF profile for users who want to use this as their default shell.
- Customize **/etc/init.options** if the shell script used for system initialization will use this shell.
- Identify the alternative shell in an input parameter in the **/etc/init.options** file.

If you are using the **/etc/profile** for the z/OS shell or **/etc/csh.login** for the tcsh shell, you may need to review them and make any necessary adjustments.

For an example of MAXPROCSYS settings in BPXPRMxx, see “Tuning process activity” on page 339.

Customizing the z/OS UNIX shells

After a user logs in to the shell, the system initializes the shell for that user. During the initialization, the system does the following:

1. Determines whether the user is authorized to use the shell by checking for a UID value in the user’s RACF user profile. It also checks that the user’s RACF group has a GID assigned to it.
2. Sets the LOGNAME, HOME, and SHELL environment variables from data in the RACF user profile, which was specified in the RACF ADDUSER and ALTUSER commands. See “Customizing the RACF user profile” on page 192.
3. Connects the user to the initial working directory that was identified in the HOME environment variable in the RACF user profile. If the RACF user profile does not identify a working directory, the system uses the root as the user’s working directory and issues a message.
4. Invokes the shell named in the PROGRAM statement of the OMVS segment in the RACF user profile.
 - a. For the z/OS shell, it is named **/bin/sh**.
 - b. For the tcsh shell, it is named **/bin/tcsh**.

Similar systems usually have an **/etc/passwd** file, which contains the HOME and PROGRAM environment variables, plus the users’ passwords. To provide better security, the z/OS shell does not use the **/etc/passwd** file; instead, it uses the initial values assigned to these variables in the RACF user profiles. RACF maintains the passwords.

See Appendix A for more information about individual environment variables.

Customizing the shell environment variables

If an environment variable is not set, it has no value. Setting an environment variable is optional.

For the z/OS shell

The places to set environment variables, in the order that the system sets them, are:

1. The RACF user profile.
2. The **/etc/profile** file, which is a system-wide file that sets environment variables for all z/OS shell users. This file is only run for login shells.
3. The **\$HOME/.profile** file, which sets environment variables for individual users. This file is only run for login shells.
4. The file named in the ENV environment variable. This file is run for both login shells and subshells.

5. A shell command or shell script.

Later settings take precedence. For example, the values set in **\$HOME/.profile** override those in **/etc/profile**.

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

For the tcsh shell

The places to set environment variables, in the order that the system sets them, are:

1. The RACF user profile
2. The **/etc/csh.login** file, which is the system-wide file that sets environment variables. This file is only run for login shells.
3. The **\$HOME/.login** file, which sets environment variables for individual users. This file is only run for login shells.
4. The **/etc/csh.cshrc** file, which is the system-wide file that sets shell variables, some environment variables (like PATH), and umask. It also defines command aliases. It is used by subshells.
5. The **\$HOME/.tcshrc** file, which sets environment variables for individual users. It is used by subshells.
6. The **\$HOME/.cshrc** file, if it is provided for compatibility with the C shell.

Later settings take precedence. For example, the values set in **\$HOME/.login** override those in **/etc/csh.login**.

Customizing the RACF user profile

The security administrator defines a user by creating a RACF user profile with an ADDUSER command or alters the user profile with an ALTUSER command. The RACF user profile contains values that are used to set the following environment variables:

LOGNAME

The TSO/E user ID

HOME The pathname of the user's home directory as specified in the HOME parameter of the RACF command. If the HOME parameter was not specified, **HOME** is the root directory. Unpredictable results may occur if an invalid or nonexistent directory is specified for the HOME parameter. For more information about setting up home directories for users, see "Defining z/OS UNIX users to RACF" on page 60.

SHELL

The pathname of the file containing the shell program as specified in the PROGRAM parameter on the RACF command. If PROGRAM was not specified, **SHELL** is **/bin/sh**.

The PROGRAM parameter can specify a special-purpose shell or another kind of program.

Customizing files for the z/OS shell

This section lists the following customization tasks:

- "Customizing /etc/profile" on page 193
- "Customizing \$HOME/.profile" on page 197

- “Customizing /etc/init.options” on page 198
- “Customizing /etc/rc” on page 201

Customizing /etc/profile

The **/etc/profile** file is the system-wide profile for the z/OS shell users. It contains environment variables and commands used by most shell users. To improve system performance, use **STEPLIB=none**.

Figure 43 shows suggested settings for **/etc/profile** provided in the IBM-supplied **/samples/profile**:

```
if [ -z "$STEPLIB" ] && tty -s;
then
    export STEPLIB=none
    exec sh -L
fi
TZ=EST5EDT
export TZ

LANG=C
export LANG

readonly LOGNAME
PATH=/bin
export PATH

LIBPATH=/lib:/usr/lib:.
export LIBPATH

NLSPATH=/usr/lib/nls/msg/%L/%N
export NLSPATH

MANPATH=/usr/man/%L
export MANPATH

MAIL=/usr/mail/$LOGNAME
export MAIL

umask 022
```

Figure 43. Partial contents of IBM-supplied **/samples/profile** (Part 1 of 2)

```

# Start of c89/cc/c++ customization section
#
# High-Level Qualifier "prefixes" for data sets used by c89/cc/c++:
#
# C/C++ Compiler:
# -----
# export ${_CMP}_CLIB_PREFIX="CBC"
#
# Prelinker and runtime library:
# -----
# export ${_CMP}_PLIB_PREFIX="CEE"
#
# z/OS system data sets:
#
# export ${_CMP}_SLIB_PREFIX="SYS1"
#
# For _CMP in _C89 _CC CXX; do
#
# Compile and link-edit search paths:
#
#   Compiler include file directories:
#
#   export ${_CMP}_INCDIRS="/usr/include /usr/lpp/cbclib/include"
#
#   Link-edit archive library directories:
#
#   export ${_CMP}_LIBDIRS="/lib /usr/lib"
#
#
# Esoteric unit for data sets:
#
# Unit for (unnamed) work data sets:
#
#   export ${_CMP}_WORK_UNIT="SYSDA"
#
# done; unset _CMP
#
# End of c89/cc/c++ customization section
#

```

Figure 43. Partial contents of IBM-supplied `/samples/profile` (Part 2 of 2)

The commands in `/samples/profile` have the following meaning:

STEPLIB=none

Indicates that STEPLIBs should be not propagated. Running the shell with STEPLIB=none assumes that the Language Environment run-time library resides in LINKLIST or in LPA.

exec sh -L

Reruns the SHELL command in the current address space with the environment variables just defined. Both **STEPLIB=none** and **exec sh -L** are run only on the first invocation of an interactive shell. The **tty -s** test prevents the shell from being run by noninteractive invocations—for example, those started with the BPXBATCH and OSHELL utilities.

TZ=EST5EDT

Sets the time zone as appropriate. In the sample profile, TZ is set to EST5EDT, which is Eastern Daylight Time.

PATH=/bin

Sets a default command search path to search only the `/bin` directory.

LIBPATH=/lib:/usr/lib:.

Specifies the directory to search for a dynamic link library (DLL) filename. If this is not set, only the working directory is searched.

NLSPATH=/usr/lib/nls/msg/%L/%N

Sets the path for message catalogs.

MANPATH=/usr/man/%L

Sets the path for the man pages.

LANG=C

Specifies the name of the default locale. C specifies the POSIX locale and Ja_JP specifies the Japanese locale.

MAIL=/usr/mail/\$LOGNAME

Sets the name of the system mail file and enables mail notification.

export TZ PATH NLSPATH MAIL LANG

Exports the values so the system will have access to them.

umask 022

Sets the default file creation mask. In the sample, the mask is set to 022. This 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.

readonly LOGNAME

Sets the LOGNAME variable to read-only so that it is not accidentally modified.

Examples of customizing /etc/profile: Copy **/samples/profile** to **/etc/profile**. If you already have **/etc/profile**, then compare it to **/samples/profile** and retrofit any new updates. Edit **/etc/profile** to change and add environment variables. For example:

- Edit the TZ environment variable to identify the time zone used by most of your users. It gives the standard time zone, the number of hours offset from Universal Time Coordinated (UTC)—also called Greenwich Mean Time (GMT)—and the daylight savings time zone. For a system with most users in New York and Boston, this variable is:

TZ=EST5EDT

For a system with most users in Houston, this variable is:

TZ=CST6CDT

- Edit the PATH environment variable to define the default command path. This variable should name all directories the installation plans to put utilities and program products in.

If you plan to place all standard utilities in the **/bin** directory, this variable is:

PATH=/bin

If you want your users to access another product's binaries which have been installed into **/usr/lpp/xxxxxxx/bin**, the variable becomes:

PATH=/bin:/usr/lpp/xxxxxxx/bin

The order of the directories in the PATH environment variable controls the search order.

If you want to add the working directory to the search, add a colon and a period (:.), as follows:

PATH=/bin:/usr/lpp/xxxxxxx/bin:.

To search the working directory first, specify:

```
PATH=./bin:/usr/lpp/xxxxxxx/bin
```

- Add the definition of the FPATH environment variable to point to the directories in **/etc/profile** that contain the shell function definitions.

FPATH contains a list of directories that the z/OS shell searches to find shell functions. Directories in this list are separated by colons. Every directory is searched in the order specified in the list until a matching function definition file is found.

That is, if you have shell functions that you want to make available to all users, define a directory that is readable by all users, and put the shell function definitions in files within this directory. Add the definition of the FPATH environment variable to **/etc/profile**. Be sure to mark it as an exported variable with the **export** command.

For example, suppose that you have a function named `buildapp` contained in a file named **/usr/lib/appdev/functions/buildapp**. You could add the following statement to **/etc/profile**:

```
export FPATH=/usr/lib/appdev/functions
```

The user can then just issue `buildapp`. The first time `buildapp` is run, it will be found in FPATH, defined in the current shell, and executed. After that first time, every time `buildapp` is issued (within the same shell), the shell executes `buildapp` without first searching for that function.

The **/samples/profile** file does not have a default FPATH setting.

FPATH follows the same format as the PATH environment variable.

- Edit the NLSPATH environment variable to specify the path that the messaging services will use to find a message catalog.
- Edit the LANG environment variable to contain the default locale name. See *z/OS C/C++ Programming Guide* and *z/OS UNIX System Services User's Guide* for more information on setting and changing the default locale.
- Edit the MAIL environment variable to define the name of the system mail file and enable notification of mail. If you plan to use a mail file other than **/usr/mail**—for example, **/usr/notes**—set the variable as follows:

```
MAIL=/usr/notes/$LOGNAME
```

- Edit the STEPLIB environment variable to define libraries that should be searched to load MVS load modules. Normally, installations should specify STEPLIB=NONE to prevent the propagation of STEPLIBs. If a STEPLIB environment variable is needed, specify only the required library.

Example:

```
STEPLIB=CEE.SCEERUN:CEE.SCEERUN2
```

If you do not specify the STEPLIB environment variable, STEPLIBs are propagated from the user's TSO/E user ID. Specifying a value other than STEPLIB=NONE can affect performance for the following reasons:

- Each time a fork or exec is invoked, STEPLIB data sets are dynamically allocated for the user.
- Each time an MVS load module is loaded, the STEPLIB data set directories are searched.
- Each time an MVS load module is found in the STEPLIB concatenation, the module is loaded from there into the user's private area storage.

- The **LOCPATH** environment variable tells the **setlocale()** function the name of the directory from which to load locale object files. If **LOCPATH** is not defined, the default directory **/usr/lib/nls/locale** is searched.

LOCPATH is similar to the **PATH** environment variable. It contains a list of HFS directories separated by colons.

For detailed information on how **setlocale()** searches for locale object files, see the description of **setlocale()** in *z/OS C/C++ Run-Time Library Reference*.

Customizing **\$HOME/.profile**

The optional **\$HOME/.profile** file contains commands that set or change the values of environment variables for an individual user. (**HOME** is a variable for the pathname for a user's working directory.) The values set in **\$HOME/.profile** can override those in **/etc/profile**. Figure 44 shows the IBM-supplied **/samples/profile**.

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

PATH=$PATH:$HOME:

EDITOR=ed

PS1='$LOGNAME': '$PWD': ' >'

export PATH EDITOR PS1
```

Figure 44. Contents of **/samples/profile**

Use the **cp** shell command to copy **/samples/profile** to your **\$HOME** directory, making the file into **.profile**. Then edit **.profile** to change and add environment variables.

ENV	Specifies the name of the user's environment file, which is a shell script. ENV=\$HOME/.setup specifies a file called .setup , which the user added to the home directory.
STEPLIB	Specifies STEPLIBs for individual users who have STEPLIB requirements that are different from those of other users. Recommendation: Use STEPLIB=none . However, there may be cases in which a specific library is needed; for example, STEPLIB=USER1.MY.USERLIB .
PATH	Appends your home directory to the current path. When you set up your own \$HOME/.profile as superuser, specify the /usr/sbin directory in your PATH variable because some superuser utilities are in that directory instead of the /bin directory. Those utilities include automount , inetd , lm , mknod , ocsconfig , rlogind , uucpd , chroot , and cron .
PS1	Specifies the prompt character or string.
TZ	Specifies a different time zone if the user is in a remote location.
MAILRC	Specifies the name of the user's mail startup file. The default is \$HOME/.mailrc .
MAIL	Specifies the name of the user's file for mail that the user does not save in some other file. The default is \$HOME/mbox .
MBOX	Specifies the name of the user's file for mail that the user does not save in some other file. The default is \$HOME/mbox .

DEAD Specifies the name of the user's file for partial messages when an interrupt or error occurs when creating a message or delivering it. The default is **\$HOME/dead.letter**.

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**.

See *z/OS UNIX System Services User's Guide* for more information about **\$HOME/.profile**.

Using an ENV environment variable file: When the shell is invoked as a login shell, **/etc/profile** and **\$HOME/.profile** are used. If users want environment variables in a shell invoked from the current shell, they should identify a file in an ENV environment variable and place shell commands in the file. The system interprets the file named in **ENV** each time a new shell is invoked.

Using a shell command or shell script: For the current shell invocation, a user can enter a shell command to set the value of any environment variable. Any variables set in a shell script are set only while the script is running and do not affect the shell that invoked the shell script (unless the script is "sourced" by running it with the dot command).

Customizing /etc/init.options

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. When this shell script finishes or when a time interval established by **/usr/sbin/init** expires, kernel services become available for general batch and interactive use.

Standard output (stdout) and standard error output (stderr) are redirected to **/etc/log**.

The files associated with the system initialization utility **/usr/sbin/init** are as follows:

/bin/sh

Default shell that **/usr/sbin/init** invokes to execute **/etc/rc** or another shell script specified in **/etc/init.options** file

/etc/init.options

Initialization options file, which is read by **/usr/sbin/init**

/etc/rc Default initialization shell script

/etc/log

The file that output is written to

Other utilities

Services that are called by the initialization shell script

/usr/sbin/init and the customized **/etc/init.options** and **/etc/rc** are run at IPL. There is no other way to invoke them explicitly.

Before **/usr/sbin/init** invokes the shell to execute the system initialization shell script, it reads the file **/etc/init.options** for values of various options. The IBM-supplied default is in **/samples/init.options**. Copy this file to **/etc/init.options** and make the appropriate changes. If you already have **/etc/init.options**, then compare it to **/samples/init.options** and retrofit any new updates. **/usr/sbin/init** treats all lines in **/etc/init.options** that do not start with a hyphen (-) as comment lines. Lines that start with a hyphen are used to specify options. The format of lines specifying options is as follows:

```
-oo vvvvv comment
```

where:

- oo is a field of one or more nonblank characters immediately following the hyphen that identify the option. The end of the option field is delimited by one or more blanks.
- vvvvv is a field of one or more nonblank characters that specify an option value. These characters are numeric, alphabetic, or a combination of both, depending on the option being specified. The end of the value field is delimited by one or more blanks.

Option and option value characters must appear in columns 1 through 79 of an option line in **/etc/init.options**. **/usr/sbin/init** ignores characters beyond column 79. However, a backslash (\) immediately following nonblank value field characters is recognized as a continuation character. If the continuation character is found, nonblank characters at the beginning of the next line are treated as option value characters. The first blank character delimits the end of the value field.

Option value characters on a continuation line are limited to columns 1 through 79.

The continuation character is recognized on continuation lines as well as the option line.

- Any characters after a blank delimiting the end of the option value field on the same line are treated as comment characters.

Options and option value ranges are listed below:

-a nnnn

Alarm option: *nnnn* are digits that specify the maximum time in seconds **/usr/sbin/init** will wait for execution of the initialization shell script to complete.

Default: 9999 seconds

Maximum: 9999 seconds

If the shell does not signal completion of the script before this time elapses, **/usr/sbin/init** writes the timeout error message, FSUM4013I, in **/etc/log** and exits with status indicating: "Timeout waiting for shell script to complete."

You must specify enough time for the system initialization script to complete if this is a requirement at your installation.

Note: If the value 0 is specified for the alarm option, no timeout interval is set. The decision to specify the value 0 for the alarm option should be made carefully and only after it is known that the initialization script is error-free.

-t *n* Terminate option: *n* is a digit indicating whether to terminate execution of the initialization shell script if the timeout specified by the alarm option (**-a**) occurs.

<i>n</i> = 0:	Allow the shell script to continue
<i>n</i> not = 0:	End the shell script
Default:	<i>n</i> = 1 (terminate the shell script)
Maximum:	1 digit

If you specify terminate and the timeout waiting for the initialization shell script occurs, **/usr/sbin/init** sends a stop signal to the shell process group.

It is your responsibility to decide if the initialization shell script can continue concurrent with batch and interactive use of the shell.

-sh *pathname*

Initialization shell pathname option: *pathname* specifies the shell that **/usr/sbin/init** should invoke to run the initialization script. **/usr/sbin/init** cannot set environment variables for the rest of the system.

Default:	/bin/sh
Maximum length:	255 characters

The line **-sh *pppp*** in **/etc/init.options** specifies the first four characters of a shell pathname, *pppp*. It also indicates that the pathname is continued on the next line (starting in column 1). Comment characters can appear after **-**.

The line **-sh** <blanks in **/etc/init.options** tells **/usr/sbin/init** not to run the shell. If you select this option, **/usr/sbin/init** does not invoke the shell to execute an initialization script. Instead, **/usr/sbin/init** signals that *multiuser mode* be entered and then exits.

-sc *pathname*

Initialization script pathname option: *pathname* specifies the initialization shell script.

Default:	/etc/rc
Maximum length:	255 characters

The line **-sc *pppp*** in **/etc/init.options** specifies the first four characters of an initialization script name, *pppp*, and indicates that the pathname is continued on the next line (starting in column 1).

-e *string*

Environment variable option: *string* in the form "*name=value*" specifying the environment variable name and the value that **/usr/sbin/init** passes to the shell it invokes.

Maximum length:	255 characters

The line **-e *ssss*** in **/etc/init.options** specifies the continuation of the environment variable name or value on the next line.

/etc/init.options can contain up to 25 **-e** option lines specifying names and values for different environment variables. **/usr/sbin/init** passes the resultant environment variable array to the shell that it invokes. In turn, the shell uses this array to set up an execution environment for the initialization shell script that is appropriate for the installation. TZ is an example of an environment variable that should be considered.

These environment variables should also be set up in **/etc/profile** or **\$HOME/.profile** for each interactive user. Examples of variables that you could specify are TZ, LANG, and NLSPATH.

Following is a sample **/etc/init.options** file showing the time zone, the Japanese language, and the locale:

```
-e TZ=JST-9
-e LANG=Ja_JP
-e NLSPATH=/usr/lib/nls/msg/%L/%N
```

/etc/init opens the message catalog **fsumucat.cat** in directory **/usr/lib/nls/msg/C** unless an NLSPATH environment variable naming a different directory is specified in the **/etc/init.options** file.

For more information on environment variables for the shell, refer to *z/OS UNIX System Services Command Reference*.

For more information about national language support, see Chapter 11.

Using REXX execs as an alternative to /etc/init: You can use a REXX exec in an MVS data set as an alternative to running the **/etc/init** initialization program. To activate the REXX exec for initialization, you must specify its name on the STARTUP_EXEC statement in the BPXPRMxx parmlib member.

Customizing /etc/rc

You need to copy **/samples/rc** file to **/etc/rc**. It contains customization commands for z/OS UNIX System Services Application Services. Figure 44 shows the IBM-supplied **/samples/.profile**. Copy this file to **/etc/rc** and make the appropriate changes. If you already have **/etc/rc**, 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 such as an **rc.tcpip** script to start **tcp** daemons.

The sample **/etc/rc** file includes the set **-v -x** command, which specifies that a verbose shell command trace of **/etc/rc** is to be written to **/etc/log**. Certain comments are also commented out.

```

# Initialization shell script, pathname = /etc/rc

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

# Provide z/OS UNIX Startup Diagnostics
set -v -x

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

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

# Allow only file owner to remove files from /tmp
chmod 1777 /tmp

# Allow only file owner to remove files from /var
chmod 1777 /var

# Allow only file owner to remove files from /dev
chmod 1755 /dev

# Setup write, talk, mesg utilities
# chgrp TTY    /bin/write
# chgrp TTY    /bin/mesg
# chgrp TTY    /bin/talk
# chmod 2755   /bin/write
# chmod 2755   /bin/mesg
# chmod 2755   /bin/talk
# Performed at install in HOT7707
# Commented out in HOT6609 and performed in SAMPLIB job FOMISCHO

```

Figure 45. Contents of sample /etc/rc file (Part 1 of 2)

```

# Setup mailx utility
# No need to CHGRP /usr/mail directory
# No need to CHGRP mailx utility
# No need to CHMOD mailx to turn on SETGID

# Setup uucp utility
# chown uucp:uucpg /usr/lib/uucp
# chown uucp:uucpg /usr/lib/uucp/IBM
# chown uucp:uucpg /usr/spool/uucp
# chown uucp:uucpg /usr/spool/locks
# chown uucp:uucpg /usr/spool/uucppublic
# chown uucp:uucpg /usr/spool/uucp/.Xqtdir
# chown uucp:uucpg /usr/spool/uucp/.Sequence
# chown uucp:uucpg /usr/spool/uucp/.Status
# chown uucp:uucpg /bin/uucp
# chown uucp:uucpg /bin/uuname
# chown uucp:uucpg /bin/uustat
# chown uucp:uucpg /bin/uux
# chown uucp:uucpg /usr/lib/uucp/uucico
# chown uucp:uucpg /usr/lib/uucp/uuxqt
# chown uucp:uucpg /usr/lib/uucp/uucc
# chmod 4775 /bin/uucp
# chmod 4775 /bin/uuname
# chmod 4775 /bin/uustat
# chmod 4775 /bin/uux
# chmod 4754 /usr/lib/uucp/uucico
# chmod 4754 /usr/lib/uucp/uuxqt
# chmod 4774 /usr/lib/uucp/uucc
# Performed at install in HOT7707
# Commented out in HOT6609 and performed in SAMPLIB job FOMISCHO

# Invoke vi recovery

# mkdir -m 777 /var/tmp
# export TMP_VI="/var/tmp"
mkdir -m 777 /etc/recover
/usr/lib/exrecovery

# Create TERMINFO database
# tic /usr/share/lib/terminfo/ibm.ti
# tic /usr/share/lib/terminfo/dec.ti
# tic /usr/share/lib/terminfo/wyse.ti
# tic /usr/share/lib/terminfo/dtterm.ti
# commented tic out in HOT1180 - all TERMINFO files are shipped

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

sleep 5
echo /etc/rc script executed, date

```

Figure 45. Contents of sample /etc/rc file (Part 2 of 2)

The setup section for the **mailx** utility has been commented out in the **/etc/rc** file because the **mailx** utility no longer requires this.

The setup section for creating the terminfo database has been commented out in the **/etc/rc** file because IBM ships the individual files that make up the terminfo database.

The setup section for the **mesg**, **talk**, **write** and **uucp** utilities has been commented out in the **/etc/rc** file because this customization is now done when running the FOMISCHO sample job.

Customizing files for the tcsh shell

This section lists the following customization tasks:

- “Customizing /etc/csh.login”
- “Customizing \$HOME/.login” on page 205
- “Customizing /etc/csh.cshrc” on page 205
- “Customizing \$HOME/.tcshrc” on page 205
- “Customizing /etc/complete.tcsh” on page 205

Customizing /etc/csh.login

The **/etc/csh.login** file is used for setting environment variables such as TERM and is only read by tcsh when it is a login shell.

Important

Because **/etc/csh.login** is the tcsh equivalent to **/etc/profile** for **sh**, you need to keep system-wide information for both sets of users in synch. Any customization that you have done for **/etc/profile** (such as setting environment variables) needs to be duplicated in C-shell syntax in **/etc/csh.login**. Future changes to **/etc/profile** also need to be made to **/etc/csh.login**. If you maintain a non-z/OS UNIX system, you could consider porting **/etc/csh.cshrc** and **/etc/csh.login** from that system to z/OS and merging them with the z/OS samples.

Figure 46 shows a sample **/samples/csh.login** file:

```
tty -s
set tty_rc=$status
if (($?STEPLIB == 0) && ($tty_rc == 0)) then
    setenv STEPLIB none
    exec tcsh -l
endif
unset tty_rc

setenv TZ EST5EDT
setenv LANG C
setenv LIBPATH /lib:/usr/lib:.
setenv MAIL /usr/mail/$LOGNAME

# =====
# Start of c89/cc/c++ customization section
# =====
# foreach _CMP( _C89 _CC _CXX)
#   setenv ${_CMP}_CLIB_PREFIX "CBC"
#   setenv ${_CMP}_PLIB_PREFIX "CEE"
#   setenv ${_CMP}_SLIB_PREFIX "SYS1"
#   setenv ${_CMP}_INCDIRS "/usr/include /usr/lpp/cbclib/include"
#   setenv ${_CMP}_LIBDIRS "/lib /usr/lib"
#
# Esoteric unit for data sets:
#   setenv ${_CMP}_WORK_UNIT "SYSDA"
# end
# unset _CMP
#
# =====
# End of c89/cc/c++ customization section
# =====
```

Figure 46. Partial contents of IBM-supplied **/samples/csh.login**

Use the **cp** command to copy **/samples/csh.login** to **/etc/csh.login**. Edit **/etc/csh.login** to change or add environment variables.

Customizing \$HOME/.login:

To change or add environment variables such as **TERM** that are customized for individual users, first use the **cp** command to copy **/samples/.login** to **\$HOME/.login**. Then edit the file to change or add environment variables. The **\$HOME/.login** file is only read by **tcsh** when it is a login shell.

Customizing /etc/csh.cshrc

The **/etc/csh.cshrc** file is the system-wide profile for **tcsh** shell users and is read by subshells.

Figure 47 shows suggested settings for **/etc/csh.cshrc** provided in the IBM-supplied **/samples/csh.cshrc**:

```
# =====
# path shell variable
# =====
#
# Specifies the list of directories that the system searches for an
# executable command.
set path = (/bin)
# =====
#
# umask variable
#
umask 022
# =====
```

Figure 47. Partial contents of IBM-supplied **/samples/csh.cshrc**

Use the **cp** command to copy the **/samples/csh.cshrc** file to **/etc/csh.cshrc**. Then edit **/etc/csh.cshrc** to change or add shell variables.

Customizing \$HOME/.tcshrc:

The **\$HOME/.tcshrc** file contains commands that set or change the values of shell variables for individual users and is read by subshells. **HOME** is a variable for the pathname for a user's home directory. The values set in **\$HOME/.tcshrc** override those in **/etc/csh.cshrc**.

Use the **cp** command to copy **/samples/.tcshrc** to your **\$HOME** directory. Then edit the new file to change or add shell variables.

Customizing /etc/complete.tcsh

The **/etc/complete.tcsh** file contains programmed completions that may be useful to the user. Programmed completions associate specific types of completion with individual commands. Use the **cp** command to copy **/samples/complete.tcsh** to **/etc/complete.tcsh**. Then edit the new file.

Configuration files

In order to use z/OS UNIX shells and utilities, you must copy the configuration files listed in Table 13 on page 206 to the specified directory.

Table 13. Configuration files

Utility	Copied	
	From:	To:
cron	/samples/queuedefs	/usr/lib/cron/queuedefs
file	/samples/magic	/etc/magic
inetd	/samples/inetd.conf	/etc/inetd.conf
lexx	/samples/yylex.c	/etc/yylex.c
mailx	/samples/mailx.rc	/etc/mailx.rc
make	/samples/startup.mk	/etc/startup.mk
OHELP	/samples/ohelp.ENU	/etc/OHELP/ENU
sh	/samples/profile	/etc/profile
	/samples/.profile	\$HOME/.profile
tcsh	/samples/complete/tcsh	/etc/complete.tcsh
	/samples/csh.cshrc	/etc/csh.cshrc
	/samples/csh.login	/etc/csh.login
	/samples/.tcshrc	\$HOME/.tcshrc
	/samples/.login	\$HOME/.login
uucp	/samples/Devices	/usr/lib/uucp/Devices
	/samples/Dialers	/usr/lib/uucp/Dialers
	/samples/Dialcodes	/usr/lib/uucp/Dialcodes
	/samples/Permissions	/usr/lib/uucp/Permissions
	/samples/Systems	/usr/lib/uucp/Systems
yacc	/samples/ypyparse.c	/etc/ypyparse.c

Installing books and a bookshelf for the OHELP command

If you have installed books and a bookshelf, you can use the OHELP command to display online information, such as online help for shell commands, shell messages, callable services, and C functions.

Instructions for installing books and a bookshelf are in *z/OS Migration*.

Enabling the man pages

To enable the man pages, do the following:

- Allocate the BookServer data set SEPHTAB and catalog it in your system.
- If the default data set 'EPH.SEPHTAB' is not used, copy the sample EPHWP00 parmlib member in SEPHSAMP into SYS1.PARMLIB.

The sample EPHWP00 parmlib member contains one line of left-justified text, "EPH". This is the IBM-supplied prefix for the SEPHTAB data set. If you change this prefix, you must change this left-justified statement to match the new prefix.

If you rename the SEPHTAB data set to another suffix, the first line of **/etc/booksrv/bookread.conf** should contain:

DSN=fully.qualified.dsn.where.members.are

The preferred location of the BookRead configuration file is **/etc/booksrv/bookread.conf**. If that is not found, **'sys1.parmlib(ephwp00)'** will continue to be used.

Setting up for mesg, talk, write, and UUCP

The customization required for the **mesg**, **write**, and **talk** utilities is done at installation time. Likewise, part of UUCP customization is done at installation time. For more information, see “Security requirements for ServerPac and CBPDO installation” on page 86. In the past, these tasks were done with the FOMISCHO job from SYS1.SAMPLIB, and prior to OS/390 2.9.0, they were done by **/etc/rc**.

The FOMISCHO job remains available in SYS1.SAMPLIB for installations that cannot synchronize their security databases for the required user ID *uucp* and group IDs *uucpg* and *TTY*. To complete this customization step, these installations must run FOMISCHO against each system image.

Customizing c89, cc, and c++ (cxx) compilers

The **c89** utility is customized by setting environment variables. The ones that most commonly require setting are specified in the **c89** customization section.

- For the z/OS shell, the customization section is in **/etc/profile**.
- For the tcsh shell, the customization section is in **/etc/csh.login**

z/OS UNIX System Services Command Reference lists the rest of the variables that might require setting for typical **c89** usage.

z/OS UNIX System Services Command Reference, in its **c89** section, assumes that the current level of z/OS C/C++ compiler and Language Environment run-time library will be used. If you must use a previous level of the compiler, then you must customize other environment variables, which are only documented here.

The environment variables used by the **cc** 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 “for loop” at the bottom of the **c89** customization section can be used. This “for loop” sets the variables for all the **c89/cc/c++** utilities.

By putting any customization statements for **c89** into **/etc/profile** for the z/OS shell (or **/etc/csh.login** for the tcsh shell) and uncommenting those lines, the environment variables are automatically exported for **cc** and **c++**. See “Customizing /etc/profile” on page 193 and “Customizing /etc/csh.login” on page 204 for more information.

After you customize the profile, you probably will not need to change it again. However, you can change the variables at any time; the next time a user logs into the shell, they will get the new settings.

Using non-default high-level qualifiers

If any of the following installed products did not use the installation default for the high-level qualifier, then the appropriate environment variable must be exported to make **c89** aware of this. The environment variables in this table are set to the default values for the current level of z/OS, but you will need to set them to your high-level qualifiers.

Note: These high-level qualifiers are used to construct the names of data sets used by **c89**. All named data sets used by **c89** must be cataloged.

Element	c89 Environment Variable
C/C++ compiler	_C89_CLIB_PREFIX=CBC
Language Environment	_C89_PLIB_PREFIX=CEE
BCP	_C89_SLIB_PREFIX=SYS1

Using a system that does not have UNIT=SYSDA

If the system is not configured with an esoteric unit SYSDA, or some other esoteric unit is to be used for VIO temporary unnamed work data sets set by **c89**, the 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:

Element	c89 Environment Variable
All c89-allocated work data sets	_C89_WORK_UNIT=SYDA

Selecting C/C++ compilers

This section lists the compiler choices; 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/profile** file that is shipped with each release. The environment variables for:

- **c89** begin with the prefix **_C89**
- **cc** begin with the prefix **_CC**
- **c++** begin with the prefix **_CXX**

If the C/C++ Class Library DLLs are used in building your executables (the default for the **c++** utility), then this will also target your executable for the same level of C/C++ Class Library

Using the same compiler for the entire system

If you are using the same compiler for the entire system, then put the compiler data set name in the linklist. By default, the linklist contains the name of the default compiler

If you are using a compiler that is not the system-wide default, then you must specify the compiler data set name in the STEPLIB environment variable and export it. Performance may be somewhat affected.

Setting up c89 to work with the current C/C++ compiler

These are the export statements for each compiler version, assuming that the default high-level qualifiers are being used. Where the **c89** environment variables are shown, the environment variables for **c++** and **cc** must also be set.

- For the current z/OS C/C++ compiler:
 - If you are using the z/OS shell, issue the following command:
export STEPLIB="CBC.SCCNCMP"
 - If you are using the tcsh shell, issue the following command:
setenv STEPLIB "CBC.SCCNCMP"

Because the current z/OS C/C++ compiler supports compiling code that is to run on previous releases of z/OS, you do not need to use any additional **c89** environment variables. All you need to do is specify the **c89** option `-Wc,"target(LEVEL)"`, where LEVEL is the level of z/OS on which the program is to be executed. For more information see the description of the TARGET option in the *z/OS C/C++ User's Guide*.

- For the IBM C/C++ V3R2 compiler:
 - If you are using the z/OS shell, issue the following commands:


```
export STEPLIB="CBC.V3R2M0.SCBC3CMP"
export _C89_CVERSION=0x13020000
export _C89_CLIB_PREFIX=CBC.V3R1M0
```
 - If you are using the tcsh shell, issue the following commands:


```
setenv STEPLIB "CBC.V3R2M0.SCBC3CMP"
setenv _C89_CVERSION 0x13020000
setenv _C89_CLIB_PREFIX CBC.V3R1M0
```
- For the AD/Cycle® C/370™ V1R2 compiler:
 - If you are using the z/OS shell, issue the following commands:


```
export STEPLIB="EDC.V1R2M0.SEDCDCMP"
export _C89_CVERSION=0x11020000
export _C89_CLIB_PREFIX=EDC.V1R2M0)
```
 - If you are using the tcsh shell, issue the following commands:


```
setenv STEPLIB "EDC.V1R2M0.SEDCDCMP"
setenv _C89_CVERSION 0x11020000
setenv _C89_CLIB_PREFIX EDC.V1R2M0)
```

Because this compiler only supports the C language, it cannot be used with the **c++** utility.

Targeting an OS/390 release earlier than the current one

The current release of Language Environment supports creating executables that will run on previous releases of OS/390. You can use the current system to build programs to run on previous releases, but the release that the program is executed on still determines what functionality is available. The C run-time library headers will detect attempts to use new function when targeting for an older releases, for releases as far back as OS/390 UNIX V2R7. There is run-time detection of attempts to use new functions on all supported older releases.

Targeting an earlier release

When targeting an earlier release, you may need to pass the 'compat' option to the binder:

```
-Wl,compat=min
```

A convenient way to do this as part of the setup is to use `_xxx_OPTIONS` (along with the other environment variables like `_xxx_VERSION`).

Examples:

- For the z/OS shell, issue:


```
export _C89_OPTIONS="-Wl,compat=min"
```
- For the tcsh shell, issue:


```
setenv _C89_OPTIONS "-WI,compat=min"
```

If you target an OS/390 UNIX release prior to V2R7, you may also wish to change the setting of the `{_INCDIRS}` environment variable so that it does not point to `/usr/include` (the default setting for **c89** and **cc**) or `/usr/lpp/cbclib/include` (the

default setting for **c++/cxx**). If you do not, you may have symbols added to the compilation namespace that were not defined in the older release, thus causing compilation errors.

You can set `{_INCDIRS}` to a null string.

Examples: :

- For the z/OS shell:
`export _C89_INCDIRS=""`
- For the tcsh shell:
`setenv _C89_INCDIRS ""`

Or, if you have other directories that you want to be automatically searched, you can add them to `{_INCDIRS}`, as long as the default directories are not used with this environment variable.

Customizing the terminfo database

Full-screen application programs such as the **vi** editor and the **more** utility require a terminfo database. The terminfo database contains the characteristics of different terminal types that are used to run these full-screen applications.

The terminfo database is shipped as part of z/OS UNIX System Services Application Services. The database is populated with the terminal types defined by **ibm.ti**, **dec.ti**, **wyse.ti**, **ansi.ti**, and **dtterm.ti**. The database is in the directory **/usr/share/lib/terminfo** and the source files are in **/samples**.

If you have been using OS/390 V2R6 or earlier, you will need to comment out the **tic** commands from your customized copy of **/etc/rc** as shown in Figure 45 on page 202.

If you need to define other terminal or workstations for a terminfo database, see “Steps for defining terminals or workstations for a terminfo database.”

Steps for defining terminals or workstations for a terminfo database

Perform the following steps to define terminals or workstations for a terminfo database.

1. Create a subdirectory in your home directory for the terminfo database terminal definition.

Example:

```
mkdir /u/myhome/terminfo
```

where *myhome* is the name of the home directory.

-
2. Copy the **.ti** file for the terminal that you are building the terminfo database for into the directory that you just created. You can obtain the terminal file from another UNIX operating system, if necessary.

Example: You can copy the file **pc.ti** into the directory:

```
/u/myhome/terminfo/pc.ti
```

3. Set the TERMINFO environment variable to the directory that the terminal definitions are in. Base your choice on the shell that you want to use.

For the . . .	Then issue . . .
z/OS shell	export TERMINFO=/u/myhome/terminfo
tcsh shell	setenv TERMINFO=/u/myhome/terminfo

4. Issue the **tic** command, specifying the terminal file.

Example:

```
tic /u/myhome/terminfo/pc.ti
```

5. Set the TERM environment variable to the name of the terminal that you want to use. Base your choice on the shell that you want to use.

For the . . .	Then issue . . .
z/OS shell	export TERM=sun
tcsh shell	setenv TERM=sun

When you are done, you have defined terminals or workstations for the terminfo database.

Re-creating the terminfo database

If for some reason you need to re-create the terminfo database, use the **tic** utility. Each type of terminal that is defined has a corresponding file with the suffix **.ti**.

Examples:

1. To define an IBM terminal for the terminfo database, specify from the shell environment:

```
tic /samples/ibm.ti
```
2. To define terminal types such as VT100 and VT220, specify from the shell environment:

```
tic /samples/dec.ti
```

For information about curses, see *z/OS C Curses*.

Customizing electronic mail

The **mailx** shell command sends electronic mail between shell users on the same system.

For the z/OS shell

To enable **mailx** processing, do the following:

- Set up a system startup file, **/etc/mailx.rc**, which contains variable values and definitions common to all shell users. The IBM-supplied sample is in **/samples/mailx.rc**. Copy this file to **/etc/mailx.rc**.

- If you use a system mailbox directory other than **/usr/mail**, identify it in the \$MAIL environment variable in **/etc/profile**. See “Customizing /etc/profile” on page 193.

Users can give names to mail files using variables in **\$HOME/.profile** or they can use files with the default names. See “Customizing \$HOME/.profile” on page 197.

For the tcsh shell

To enable **mailx** processing, do the following:

- Set up a system startup file, **/etc/mailx.rc**, which contains variable values and definitions common to all shell users. The IBM-supplied sample is in **/samples/mailx.rc**. Copy this file to **/etc/mailx.rc**.
- If you use a system mailbox directory other than **/usr/mail**, identify it in the \$MAIL environment variable in **/etc/csh.login**. See “Customizing /etc/csh.login” on page 204.

Users can give names to mail files using variables in **\$HOME/.login** or they can use files with the default names. See “Customizing /etc/csh.login” on page 204.

Chapter 11. Customizing for your national code page in the shell

Overview

This chapter explains how to set up a default language for all users of the z/OS shell. It also explains how to customize your system so that z/OS UNIX messages are displayed in Japanese or Simplified Chinese. (They are available in English, Japanese, or Simplified Chinese.)

Tips:

- For the z/OS shell, if you want to set the language for yourself, or for just one user, you can make these changes in the **\$HOME/.profile**, or log on to the z/OS shell and export the LANG and NLSPATH environment variables.
- For the tcsh shell, if you want to set the language for yourself, or for just one user, you can make these changes in the **\$HOME/.login**, or log on to the z/OS shell and set the LANG and NLSPATH environment variables.

See the appendix in *z/OS UNIX System Services User's Guide* for information on the locale objects, source files, and charmaps that the UNIX System Services Application Services support.

In this chapter

This chapter covers the following subtasks.

Subtasks	Associated procedure (see . . .)
Setting up your national code page	"Steps for setting up your national code page"
Customizing for Japanese and Simplified Chinese	"Steps for customizing the login file for the z/OS shell" on page 216
	"Steps for customizing the login file for the tcsh shell" on page 216
	"Steps for displaying messages" on page 217
	"Steps for activating MVS Message Service (MMS)" on page 217

Setting up your national code page

This section provides the general setup information for setting up your national code page for shell users. If you will be using Japanese or Simplified Chinese, you still need to do these steps first before going on to "Customizing for Japanese and Simplified Chinese" on page 216.

Steps for setting up your national code page

Before you begin: You need to have the login file for your shell.

1. For the z/OS shell, copy **/samples/profile** to **/etc/profile**. You may have already done this, as described in "Customizing /etc/profile" on page 193.
2. For the tcsh shell, copy **/samples/csh.login** to **/etc/csh.login**. You may have already done this, as described in "Customizing /etc/csh.login" on page 204.

Perform the following steps to set up your national code page for shell users.

1. Customize the login file for your shell.

For this shell	Do this . . .
z/OS shell	Customize /etc/profile so that your selected national page is enabled when the shell is first invoked. Be careful that the shell, with the updated /etc/profile, does not keep restarting itself after you restart the shell. To make sure that exec sh -L is executed only once, copy the code in the sample /etc/profile and update it with your national code page.
tcsh shell	Customize /etc/csh.login so that your selected national page is enabled when the tcsh shell is first invoked. Be careful that the shell, with the updated /etc/csh.login does not keep restarting itself after you restart the shell. Tip: To make sure that exec sh -l is executed only once, you can copy the code shown in the sample /etc/csh.login, updated with your national code page.

2. Convert from ASCII to your national code page. Use the **chcp** command to change the data conversion for rlogin sessions.

- For the z/OS shell, the following sample **/etc/profile** shows examples of statements to convert the terminal session data using ASCII code page ISO8859-1 and EBCDIC code page IBM-277. This example uses the Danish locale.

```
if test -z "$LOCALE_SWITCH" && tty -s
then
  echo " - - - - - "
  echo " - Logon shell will now be invoked to reflect - "
  echo " - code page IBM-277 - "
  echo " - - - - - "
  LOCALE_SWITCH=EXECUTED
  LANG=C
  LC_ALL=Da_DK.IBM-277
  export LANG LC_ALL LOCALE_SWITCH

  # Issue chcp if not using OMVS command
  if test "$?_BPX_TERMPATH != "OMVS" ) then
    chcp -a ISO8859-1 -e IBM-277
  fi
  exec sh -L
else
  echo " - - - - - "
  echo " - Welcome to z/OS UNIX System Services - "
  echo " - - - - - "
fi
```

- For the tcsh shell, the following sample **/etc/csh.login** shows examples of statements to convert the terminal session data using ASCII code page ISO8859-1 and EBCDIC code page IBM-277. This example uses the Danish locale.


```

tty -s
set tty_rc=$status
if (($?LOCALE_SWITCH == 0 && tty_rc == 0)) then
    echo " - - - - - "
    echo " - Logon shell will now be invoked to reflect - "
    echo " - code page IBM-277 - "
    echo " - - - - - "
    setenv LOCALE_SWITCH=EXECUTED
    setenv LANG=C
    setenv LC_ALL=Da_DK.IBM-277
    # Issue chcp if not using OMVS command
    if ($?_BPX_TERMPATH != "OMVS" ) then
        chcp -a ISO8859-1 -e IBM-277
    endif
    exec tcsh -l
endif
unset tty_rc

```

-
3. Convert these files to your selected locale, using the **iconv** command.

```

/etc/yylex.c
/etc/mailx.rc
/etc/startup.mk
/etc/yparse.c

```

The **lex**, **mailx**, **make**, and **yacc** utilities expect both system files and user files to be in the same code page.

Example:

```
iconv -f IBM-1047 -t IBM-277 /etc/mailx.rc >/etc/mailx.rc.227
```

-
4. Update BPXBATCH or OSHELL, if necessary.

Tip: If you use BPXBATCH or OSHELL (which uses BPXBATCH), you must do this step in order to get the code page working immediately under BPXBATCH and OSHELL. Use the STDENV ddname to point to a file or data set that contains the environment variable definitions for the code page. The code page you specify will not affect the shell because ddname is read before the first shell is invoked, (Because the STDENV DD statement does not affect the OMVS command, you need to put the environment variables in **/etc/profile**.)

For more information about BPXBATCH and STDENV, see *z/OS UNIX System Services User's Guide*.

-
5. If you need to customize for Japanese or Simplified Chinese, go to "Customizing for Japanese and Simplified Chinese" on page 216.

-
6. If you do not need to customize for Japanese or Simplified Chinese, save the login file.

- For the z/OS shell, it is **/etc/profile**.
 - For the tcsh shell, it is **/etc/csh.login**.
-

When you are done, you have set up your national code page.

Tip: To verify your code page, issue:

```
echo $HOME
```

If you entered the shell before the code page was set up, you will see \$HOME. Otherwise, the shell will display the pathname of your home directory. The \$ should be read as your code page's dollar sign.

Customizing for Japanese and Simplified Chinese

If you are customizing for Japanese or Simplified Chinese, you need to make more changes to your login file after completing the steps in "Setting up your national code page" on page 213. If you want to display your messages in Japanese or Simplified Chinese, you need to customize **/etc/init**. These changes take effect the next time OMVS is started.

Tip: You can set the system default to display translated messages. "Steps for activating MVS Message Service (MMS)" on page 217 describes the procedure.

The examples are for Japanese. Equivalent changes are required to customize for Simplified Chinese.

Steps for customizing the login file for the z/OS shell

Before you begin: You need to have **/etc/profile** set up so that you can edit it.

1. Change the line **LANG=C** to **LANG=Ja_JP**.

-
2. Add the following line:

```
LC_ALL=Ja_JP.IBM-939
```

-
3. Ensure that **LANG** and **LC_ALL** are specified on the line containing **export**.

-
4. Save **/etc/profile**.
-

When you are done, you have customized the login file for the z/OS shell so that it runs in the Japanese locals.

Steps for customizing the login file for the tcsh shell

Before you begin: You need to have **/etc/csh.login** set up so that you can edit it.

1. Change the line **setenv LANG=C** to **LANG=Ja_JP**.

-
2. Add the following line:

```
setenv LC_ALL Ja_JP.IBM-939
```

-
3. Save **/etc/csh.login**.
-

When you are done, you have customized the login file for the tcsh shell so that it runs in the Japanese locales.

Steps for displaying messages

Before you begin: You need to have `/etc/init.options` set up so that you can edit it.

1. Locate the following line:

```
*e LANG=En_US.IBM-1047
```

2. Replace it with:

```
-e LANG=Ja_JP
```

3. Locate the line:

```
*e NLSPATH=/usr/lib/nls/msg/%L/%N
```

4. Replace it with:

```
-e NLSPATH=/usr/lib/nls/msg/%L/%N
```

5. Save `/etc/init.options`.
-

When you are done, you have customized the `/etc/init` to display messages in Japanese.

Steps for activating MVS Message Service (MMS)

Perform the following steps to activate the MVS Message Service.

1. Compile the English and translated message skeletons.
-

2. Create or update the following SYS1.PARMLIB members to initialize values for MVS Message Service:

- MMSLSTxx
 - CNLcccxx
 - CONSOLxx to define the MMSLSTxx member in effect for the system
-

3. Activate MVS Message Service.

Example: One way to activate MVS is to issue `SET MMS=xx` from the MVS operator console, where `xx` refers to the MMSLSTxx member of SYS1.PARMLIB.

When you are done, you have activated the MVS Message Service; translated messages will be displayed.

Tip: Because MVS Message Service does not support translating messages to the MVS operator console, you must set up a TSO/E console that mirrors the operator's console in order to see the translated messages. TSO/E displays Japanese and Simplified Chinese messages to DBCS terminals only.

TSO/E messages

TSO/E messages are issued through MVS Message Service. For more information, see the section “Providing Translated Messages” in the chapter “Customizing TSO/E for Different Languages” in *z/OS TSO/E Customization*.

If you do not want Japanese or Simplified Chinese to be the default language, but want to see translated messages on your terminal, follow these instructions:

- For Japanese, issue **PROFILE PLANGUAGE(JPN)** at the TSO/E READY prompt on your DBCS terminal. This TSO/E command sets the primary language. The code JPN must match the LANGCODE statement in SYS1.PARMLIB(MMSLSTxx).
- For Simplified Chinese, issue **PROFILE PLANGUAGE(CHS)** at the TSO/E READY prompt on your DBCS terminal. The code CHS must match the LANGCODE statement in SYS1.PARMLIB(MMSLSTxx).

TSO/E help panels

The TSO/E help panels must be set up separately. Edit your SYS1.PARMLIB(IJKTSOxx) member in effect and ensure that the HELP statement refers to where the TSO/E help files are.

Tip: If you allocate a SYSHELP DDNAME in SYS1.PARMLIB, TSO/E searches there, rather than in the data sets pointed to by the HELP statement. For the format of the HELP statement, see *z/OS TSO/E Command Reference*.

See the section “Specifying Help Data Sets” in the chapter “Customizing TSO/E for Different Languages” in *z/OS TSO/E Customization* for more information on setting up help data sets.

Concatenating target libraries to ISPF

Rules:

- To use the Japanese translation of the panels, messages, and tables, you must concatenate the following target libraries to the appropriate ISPF data definition names (ddnames):
 - SYS1.SBPXPJPN to ISPLIB
 - SYS1.SBPXMJPN to ISPLMLIB
 - SYS1.SBPXTJPN to ISPTLIB
 - SYS1.KHELP to SYSHELP
- To use the Simplified Chinese translation, concatenate the following target libraries to the appropriate ISPF ddnames:
 - SYS1.SBPXPCHS to ISPLIB
 - SYS1.SBPXMCHS to ISPLMLIB
 - SYS1.SBPXTCHS to ISPTLIB
 - SYS1.PHELP to SYSHELP

PROFILE PLANGUAGE and the OMVS command

The PROFILE PLANGUAGE setting in effect when the OMVS TSO/E command is first issued determines the language for all OMVS command messages not from TSO/E, until you exit OMVS and return to TSO/E.

If PROFILE PLANGUAGE(JPN) is specified, and later you go to TSO/E and enter PROFILE PLANGUAGE(ENU), most TSO/E messages appear in English—including TSO/E messages about the OMVS command.

However, any OMVS command message not from TSO/E (such as the help panels invoked from <PF1> or the FSUM23-prefix messages) appear in Japanese. In particular, the TSO/E prompt message “OMVS - enter a TSO/E command” still appears in Japanese but all other messages appear in English while you are in TSO/E.

Chapter 12. Configuring the UNIX-to-UNIX copy program (UUCP)

Overview

This chapter discusses the UNIX-to-UNIX copy program (UUCP). You must decide if you want to have your system participate in a UUCP network. If you already have made that decision, go to “Configuring your local system” on page 225.

What is UUCP?

UUCP is a group of programs, directories, and files that can be used to communicate with any UNIX system that is running a version of the UNIX-to-UNIX copy program (UUCP). A UUCP network traditionally consists of a group of computers joined in a network using serial connections or TCP/IP. The UNIX System Services implementation of UUCP uses TCP/IP; it does not provide modem support.

The UUCP functions are used to automatically transfer files and requests for command execution from one UUCP system to another, typically in batch mode at scheduled intervals. You can use UUCP for file transfer, remote command execution, and custom applications.

If you use a UUCP utility to transfer a file or execute a remote command, a job request is created and queued. Depending on how UUCP has been configured at your system, the job may be processed immediately or remain queued and only be processed at scheduled times. At some point, either your system will contact the remote system, or be contacted by the remote system at which time the queued jobs will be processed. For security purposes, configuration files on each system control which transfers can take place and which commands can be executed. (See “Create or edit configuration files” on page 229.)

The UNIX System Services version of UUCP is XPG4-compliant.

Transferring files

UUCP can send and receive files between systems. The **uucp** command queues requests for file transmission or retrieval, and invokes **uucico** to establish the connection with the remote system and complete the transfer. Based on configuration specifications, file transfers with the remote system may not be allowed. The **cron** daemon can be used to invoke **uucico** to send the queued files in the background when appropriate. After **uucico** has made a connection with a remote system, and local **uucp** requests have been processed, file transfer requests created on the remote system are processed.

Executing commands from a remote location

The **uux** command allows you to run a program at another system, with the appropriate permissions. An execute file is sent to the remote system where it is treated as a command (like a batch file).

Custom applications

You can also tailor UUCP for custom applications, which can send or collect data from remote systems and execute commands remotely. A common application built on UUCP is public discussion groups, called **netnews**, or simply **news**. The net is a

public forum (consisting of member systems) for the exchange of ideas in the form of news articles. Users belonging to the member systems can post, read, and reply to news.

UUCP commands and daemons

UUCP provides the **uucp** command, which schedules files to be exchanged with other UUCP systems, and the **uux** command, which schedules commands to be executed by other UUCP systems. However, the **uucp** and **uux** commands do not cause any files to be exchanged or commands to be executed. For this, UUCP provides two daemons called **uucico** and **uuxqt** which establish communication sessions, transfer data, and execute commands according to the requests scheduled by **uucp** and **uux**.

The commands associated with UUCP are:

uucc	Compile UUCP configuration files
uucp	Copy files between remote systems
uulog	Display log information about UUCP events
uuname	Display a list of UUCP systems
uupick	Manage files sent to you via uuto
uustat	Display the status of pending UUCP transfers
uuto	Copy files to users on remote systems
uux	Request command execution on remote systems

The daemons associated with UUCP are:

cron	Starts the uucico daemon according to the schedule specified
inetd	This TCP/IP daemon starts the uucpd daemon
uucico	Processes uucp and uux file transfer requests
uucpd	Invokes uucico for TCP/IP connections from remote uucp systems
uuxqt	Run commands from other systems

UUCP directories and files

The directories associated with UUCP are:

- **/usr/spool/uucppublic**, the public UUCP directory that is the default directory for storing files that have been transferred to the local system by **uucp**.
- **/usr/spool/uucppublic/receive**, a subdirectory in the public directory for files sent from remote systems using **uuto**.
- **/usr/spool/uucp**, the spool directory that holds all work requests and all log files for UUCP.
- **/usr/spool/uucp/.Sequence**, a subdirectory for sequence files used by **uucp** and **uucico**.
- **/usr/spool/uucp/.Status**, a subdirectory containing status files for each remote system.
- **/usr/spool/uucp/.Xqtdir**, the working directory for **uuxqt**.
- **/samples**, the directory that sample configuration files are shipped in.
- **/usr/lib/uucp**, the directory that customized configuration files reside in.

For a discussion of configuration files, see “Create or edit configuration files” on page 229.

For a discussion of system files, see “Log files, lock files, status files, and working files” on page 243.

The UUCP communications network

A UUCP network consists of a number of systems that exchange information. Each system has a working copy of UUCP and a unique name that identifies it in the network. There is no central control system for a UUCP network; each system controls its own connections. In a UUCP network, computers connect computers in the same building or to networks that include computers around the world.

In a UUCP network, every system (also known as a site) communicates with at least one other system, but doesn't have to call all the sites. See Figure 48 for a diagram of a simple network. This network has four sites named North, South, East, and West.

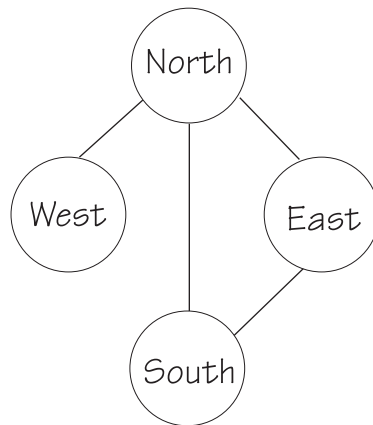


Figure 48. A simple UUCP network

The lines indicate a direct connection between two systems:

North Connects to East, South, West

East Connects to North, South

South Connects to North, East

West Connects to North

Each system exchanges files with the systems it calls directly. Users on North can send files directly to any of the other three systems, but users on West can only send files directly to North. These are called direct connections to distinguish them from connections made through intermediaries. Someone on West can send a file to someone on East indirectly through North, if North has agreed to pass along file requests from West to East. This makes North an intermediary node.

Alternatively, North could set up its configuration so that West could not transfer files through North, but only to North. This is called a *terminal* or *leaf-node* connection. For information on how to define a connection between two nodes, see the description of the `COMMANDS` option in “The Permissions File” on page 234.

Securing your system

UUCP gives users on other systems access to your computer. By default, remote users can only write data to your public directory; they cannot read any data nor can they execute any commands.

However, remote users potentially could copy files to your file system or from your file system. They could also run commands on your system. How do you ensure that they do not remove files you want, read your private files, or run commands that damage your system? In short, how do you keep your UUCP system secure?

Security considerations

There are three things to consider in security:

- Authorization—Who is authorized to access your system?
- Access—What files can users on other systems read and write?
- Execution—What commands can users on other systems run on your system?

Authorization is the highest level of security. Only those with the current NUUCP password can access your system and even then, only authorized systems can use it. There is one catch, however, and that is when more than one system is involved in the file transfer (a multi-hop transfer). If South allows North access, there is nothing South can do to prevent North from giving West the ability to use North as an intermediary node between South and West. South cannot differentiate between requests originating from North and requests being forwarded through North.

Using permissions

To deal with the security issues of access and execution, UUCP uses the concept of *permissions*. For each directly connected system, you assign access permissions to look at a specific portion of your file system and execute permission to run certain commands.

Permissions can be broad or restrictive. If you are using UUCP to connect a group of machines in your office, you may want everyone to have access to all the files and be able to run all of the commands on each machine. On the other hand, you may not want private files to be made public.

For example, imagine a central office with many branch offices. The central office uses remote commands to run reports in each branch office, and send the results back to the central office. The central office needs permission to run the command that produces reports, and it needs permissions to read and write the associated files. People on other systems don't need those files or permissions. In fact, it could be dangerous to the company to allow those permissions to anyone else.

If one of the branch offices has a connection to a different UUCP network, private information could go out worldwide. The branch office denies that outside connection permission to run any commands which produce reports or to read those files. It limits the outside system to reading and writing in a small part of the file system, perhaps one directory. This directory is the only part of the file system that all other UUCP systems can read or write — it's public. Not surprisingly, this directory is called the *public UUCP directory*.

The public UUCP directory

The public UUCP directory is the default destination for files that have been transferred to the local system from other systems. Additionally, if a remote user has read access to the local system, by default the directory he can read from is the public directory. The files remain in the public directory until users claim them.

Typically, users on the local system have read access to the public directory (and sometimes write and execute access as well). So users can access files in the public directory using normal file access methods (for example, **cp**, **cat**, or **vi**)—or for files sent by **uuto**, they can use **uupick** to handle them. The **uuto** command, a

simplified method of using **uucp**, uses the **receive** subdirectory of the public directory as its target. Within that subdirectory, each user on the local system has a subdirectory.

The public UUCP directory is called **/usr/spool/uucppublic**, and it is created when the BPXISMKD job is run as part of the z/OS installation. (Other UUCP systems or operating systems may use different names for the public directory.) Within the public directory, UUCP creates a subdirectory for each remote system that sends files to the local system.

To make file transfers easier, you can use a special character in pathnames for the public UUCP directory: when tilde (**~**) is written as the first directory in a destination path name, the **~/** stands for the public UUCP directory. You can specify the public UUCP directory with the pathname **~/**. The public UUCP directory is defined as the home directory of the user **uucp**, so you can also specify it as **~uucp**.

Execute permissions

By default, UUCP does not give permission for remote systems to run any commands; you must specify the commands that remote systems can run at the local system.

If you are willing to pass along files copied with **uucp**, or if you want to allow other direct systems to use wild cards when requesting files from your system, give the remote system permission to run the **uucp** command. If you do not assign execute permissions for the **uucp** program to another system, it can transfer files only *to* your system (meaning your system is a terminal node, or leaf-node) but not *through* your system (meaning your system is an intermediary node).

If a remote system has permission to run **uucp**, it can be an intermediary for another system to which it is connected. Your local system cannot distinguish if an incoming request from an authorized system originated at that system or at a system to which it is connected. Therefore, you must assume that the execute permission you give to a remote system can be inherited, or used, by another system to which it is connected.

To use wild cards to request files from your system, the remote system must have permission to run **uucp** and also have read permission on the directory holding the files.

- For an example of a multiple system **uucp** transfer, see the **uux** man page or the command description in *z/OS UNIX System Services Command Reference*.
- For an example of how to give a remote system permission to run **uucp**, see “Entries in the permissions file” on page 237.

Configuring your local system

To configure your local system for UUCP access, you must:

1. Determine your local system name.
2. Create or edit configuration files.
3. Define the new user ID, NUUCP, to RACF. The other required user ID, **uucp**, and group ID, **uucpg**, were already defined at installation time.

The following sections describe these tasks.

Determine your local system name

To determine what your system is called, in the shell environment type:

```
uname -n
```

This gives the name by which your system is known in a communications network; this is the name specified by the IPL parameter SYSNAME. In UNIX System Services, UUCP recognizes the first eight characters of this name (other UNIX systems may recognize more or fewer characters).

Add an entry to the permissions file

UUCP uses five different configuration files to describe various aspects of your UUCP setup. (To learn more about the configuration files, refer to “Create or edit configuration files” on page 229 before proceeding with this section.)

The Permissions file is used to control the access that remote systems have to data and programs on the local system. You may want to change some of the default settings of the Permissions file.

If you need to change some of the default permissions for your local system (such as PUBDIR, READ, WRITE, NOREAD, or NOWRITE) then you will need an additional entry in the Permissions file for your local system. If you do not need to change the default permissions then you do not need an entry in the Permissions file for your local system.

For example, if you wanted to change your uucp public directory, your Permissions file may look like this:

```
MACHINE=local          \
  READ=/readall        \
  PUBDIR=/free
MACHINE=site1:site2:SITE3 \
  READ=/readall        \
  COMMANDS=uucp:cat:cp:ls
LOGNAME=NUUCP          \
  READ=/readall        \
  PUBDIR=/free         \
  SENDFILES=yes        \
  VALIDATE=site1:site2:SITE3
```

Define the group ID and the user ID to RACF

As a customization step for UUCP, a UUCP-specific group ID (*uucpg*), and at least two user IDs are defined. The user IDs are:

- *uucp*, the user ID that owns all the UUCP files and directories. Use it when editing configuration files or performing other administrative tasks. The user ID *uucp* and group ID *uucpg* are now requirements for ServerPac and CBPDO installations. See “Security requirements for ServerPac and CBPDO installation” on page 86.
- A LOGNAME user ID that remote systems use when dialing in to your system. Traditionally, this user ID begins with NUUCP. For purposes of example here, we use NUUCP as the user ID. You may want to establish more than one LOGNAME user ID to handle different levels of access for remote systems.

The following sections describe how to define these IDs to RACF. (If you are using an equivalent security product, refer to that product’s documentation.) All the RACF commands are issued by a TSO/E user ID with RACF SPECIAL authority. Three procedures are described:

- “If you use uppercase IDs”
- “If you use mixed-case group and user IDs”
- “If you have problems with using NUUPC”

IBM highly recommends that these entries be duplicated in all of your security data bases, including the same UID and GID values in the OMVS segment. This will make it easier to transport the HFS data sets from test systems to production systems.

For more information, see “Defining z/OS UNIX users to RACF” on page 60 and “Creating z/OS UNIX groups” on page 73.

If you use uppercase IDs

If you use only uppercase IDs on your system, these are the RACF commands to define the group ID and user IDs.

1. To define the LOGNAME user ID (in this example, it is specified as NUUCP), issue:

```
ADDUSER NUUCP DFLTGRP(UUCPG) PASSWORD(A4B3C2D1)
OMVS(UID(397) HOME('/usr/spool/uucppublic')
PROGRAM('/usr/lib/uucp/uucico'))
```

where 397 is an example of a unique OMVS UID (do not use an UID of 0).

2. You may want to define other user IDs similar to NUUCP to provide different access to your systems resources to the different remote systems issuing requests to your system. Each would have a unique UID, but would have the same attributes as NUUCP. In particular, each must have home directory of **/usr/spool/uucppublic** and initial program of **/usr/lib/uucp/uucico**. The UUCP permissions file is used to specify what these user IDs can access, as explained in “The Permissions File” on page 234.

If you use mixed-case group and user IDs

If you already use mixed-case group and user IDs on your system and the users do not conflict with existing names, perform the steps for uppercase IDs in “If you use uppercase IDs.”

You may want to add the lowercase names to your alias table, mapping them to uppercase names. This is not necessary, because when the lowercase names are not found in the alias table, they are folded to uppercase. For more information about the alias table, see “USERIDALIASTABLE” on page 37.

If you have problems with using NUUPC

If a name such as NUUPC is not allowed on your system (or if it conflicts with an existing name), these are the RACF commands to define the user ID.

1. To define a LOGNAME user ID of *xxnuucp* :

```
ADDUSER xxnuucp DFLTGRP(UUCPG) PASSWORD(A4B3C2D1)
OMVS(UID(397) HOME('/usr/spool/uucppublic')
PROGRAM('/usr/lib/uucp/uucico'))
```

where 397 is an example of a unique OMVS UID (do not use an UID of 0) and *xxnuucp* is replaced by a 1-to 7-character user ID of your choice. This is the user ID that remote systems use when communicating with your system.

You may want to define other user IDs similar to NUUCP to provide different access to your system resources to the different remote systems issuing UUCP requests to your system. Each would have a unique UID, but would have the same attributes as NUUCP. Each must have home directory of

/usr/spool/uucppublic and initial program of **/usr/lib/uucp/uucico**. The UUCP Permissions file is used to specify the accessibility of each of these user IDs.

2. Set up a User ID alias table.

Tip: Using the alias table causes poorer performance as well as increases systems management costs and complexity. For more information about the alias table, see “USERIDALIASTABLE” on page 37.

3. In your User ID and Group Name alias table:

- Specify the user ID aliases:

```
xxnuucp nuucp
xxuucp uucp
```

- Specify the group alias:

```
xxuucpg uucpg
```

Configuring communication with remote systems

To configure UUCP so that it can communicate with remote systems, you must establish the appropriate communication protocols in the Systems file and create working directories for each supported remote system. To do this:

1. Obtain information about remote systems.
2. Create or edit the required configuration files.
3. Compile the configuration files with the **uucc** utility.
4. Create working directories for local and remote systems.
5. Schedule UUCP transfers with **cron**.

The following sections describe these tasks.

Obtain information about remote systems

Before attempting remote system configuration, contact the system administrator for that system. Together, you must decide if your system can call the other system, the other system can call your system, or either system can call each other.

If your system is going to call the remote system, you need the following information:

- The UUCP name of the remote system.
- A UUCP login account on that system. You need a login name and a password, so your system can log into the remote system.
- The login procedures. Ask whether any special send/expect sequences are used. (Send/expect sequences are explained in the description of chat scripts in “The systems file” on page 229.) Alternatively, you can attempt a remote login and note the sequence of prompt strings and commands required to login.
- You may need scheduling information, because there may be times when UUCP connections are not allowed. You can control the time when calls are made—for example, when lower rates are available—either with the scheduling information in the Systems file or when setting up **cron**.

If the remote system will call your system, you must provide the following information:

- Your system name.
- A login name for the remote system.
- A UUCP password for the remote system.
- The send/expect sequence. For z/OS systems, this would normally be in the following format:

```
in:--in: uucp_login_user password: password
```

where:

- **uucp_login_user** is the login user ID that the remote system is authorized to use—such as NUUCP.
- **password** is the password for the login user ID.

Create or edit configuration files

UUCP uses five different configuration files to describe various aspects of your UUCP setup. Sample configuration files are shipped in the **/samples** directory. You can customize them. To do this, log in as the UUCP user or use the **su** command to switch to the user **uucp** so that you can create or change configuration files. Copy each configuration file into the **/usr/lib/uucp** directory and customize it with entries for your local system. Table 14 lists these files and summarizes the information each contains.

Table 14. UUCP configuration files

File	Contents
Systems	Lists each supported system and describes when and how to establish a connection for each system
Devices	Describes communications hardware on your system.
Dialers	Contains dialing instructions for your system's modems
Dialcodes	Defines abbreviations that can be used as part of phone numbers.
Permissions	Defines for each remote system the sections of your file system that can be read from or written to and the commands which can be executed on your system by that system. Also defines how your system exchanges queued work with remote systems.

If you need to make changes to your configuration, edit the configuration files and then run the **uucc** command to compile them.

Editing a configuration file

If you need to extend an entry in the configuration file over two or more lines, end all lines, except the last one, with a backslash (\). Look at this Permissions file, for example:

```
LOGNAME=uwest MACHINE=west READ=/ WRITE=/      \  
  COMMANDS=uucp:cat NOREAD=/usr/private \  
  NOWRITE=/usr/private SENDFILES=yes REQUEST=yes \  
  VALIDATE=west
```

```
LOGNAME=nuucp MACHINE=OTHER REQUEST=yes \  
  SENDFILES=call
```

The systems file

The Systems file contains at least one entry for each remote system that your system is going to call. It provides information for the **uucico** utility to use when it is invoked. Each entry in the file has the format:

```
system sched device_type speed phone chat_script
```

For example,

```
sys2 Any TCP - sys2.kgn.ibm.com in:--in: nuucp ssword: uupasswd
```

The following list describes what each field represents:

system

The name of a remote system. This name must be unique (compared to other remote system names in the **Systems** file) in its first seven characters.

sched The times when your local system is permitted to call *system*. There are four subfields in the sched field: day, time, grade, and retry.

An example sched field looks like this:

Mo1200/C;5

where Mo is the day subfield, 1200 is the time subfield, /C is the grade subfield, and 5 is the retry subfield.

The description of each subfield follows:

day Indicates which days of the week your system can call the remote system named by *system*. The abbreviations Mo, Tu, We, Th, Fr, Sa, and Su represent individual days. You can also use the following keywords:

Any Your system can call the remote system on any day.

Never Your system should never call the remote system. It should only wait to be called.

Wk Your system can call the remote system on any weekday (that is, Monday-Friday).

time The range of times during which your system can call the remote system named by *system*. This subfield immediately follows the day subfield with no intervening spaces. The times given apply only to days specified by day. If you do not specify a time subfield, your system can call the remote system any time during the given days. The format of this subfield is:

time1-time2

where both time1 and time2 are 24-hour clock times. For example,
WeTh0730-1415

means that your system can call the remote system between 7:30 a.m. and 2:15 p.m. on Wednesdays and Thursdays. This time range can extend over 0000 (midnight), but be careful. It doesn't quite work the way you might expect it to. For example,

Mo2300-0700

does not indicate 11:00 p.m. on Monday[™] through 7:00 a.m. on Tuesday, but rather midnight through 7:00 a.m. on Monday morning and 11:00 p.m. through 11:59 p.m. on Monday evening.

You can specify multiple day/time combinations in an entry by separating them with a comma. For example a **Systems** file entry containing

Th0800-1600,Fr1215-1900,SaSu

indicates your system can call the remote system during the following times:

8:00 a.m. through 4:00 p.m. on Thursday

12:15 p.m. through 7:00 p.m. on Friday

Anytime on Saturday or Sunday

grade An optional subfield that lets you specify the minimum grade of work file that **uucico** will send during a given time period (as indicated by the day/time subfields). A grade is a single digit, or a single uppercase or lowercase letter. In order of priority, from highest to lowest, the grades are arranged

0 1 2 ... 9 A B ... Z a b ... z

That is, 0 has the highest priority and z has the lowest).

As work files are created for UUCP file transfers, they are automatically assigned grades that determine the order in which they are sent. By default, **uux** requests have a grade of A and **uucp** requests have a grade of n.

This optional subfield is separated from a day/time pair by a slash. For example,

MoTu0800-1200/C

indicates that only work files with a grade of C or higher will be sent during the hours of 8:00 a.m. to noon on Mondays.

The grade subfield only controls outgoing files during the given time period. It does not affect incoming files.

retry An optional subfield that indicates how many minutes after an unsuccessful call to a remote system, **uucico** should wait before trying to call that system again. This subfield, if specified, appears at the end of the sched field (separated by a semicolon). For example, a sched field of

Any;60

indicates that your system can try to call the remote system at any time and if it is not successful in connecting, it will not try again for 60 minutes.

If you do not include the retry subfield, **uucico** waits five minutes after the first unsuccessful connection attempt. This waiting period doubles after each subsequent failure.

device_type

Only TCP/IP connections are supported, so specify TCP.

speed Only TCP/IP connections are supported, so specify – (hyphen).

phone Only TCP/IP connections are supported, so this field must contain the IP address of the remote system, or a host name by which the IP address is known. You should be able to ping this address. For example, from TSO/E:

```
ping omvsoe2a
PING V3R1: PINGING HOST OMVSOE2A (198.151.241.130). USE
PING: PING #1 RESPONSE TOOK 0.004 SECONDS. SUCCESSES SO FAR
```

chat_script

A text string that defines the initial login conversation that takes place between your system and the remote system. It has the format:

expect_string send_string expect_string send_string ...

where *expect_string* is the text string that you expect to receive from the remote system and *send_string* is the text_string that you want to send in response. These two strings are separated by blanks. For example, when you login to a remote system, it responds with

login:

Type

nuucp

and press ENTER. The remote system then replies

password:

Enter your password

Shazam!

and press ENTER.

This conversation can be expressed as the following chat script:

```
login: nuucp password: Shazam!
```

This chat script tells **uucico** to expect the string *login*. After it is received, reply by sending the string *nuucp* (automatically sending a newline afterwards). **uucico** then waits for the string *password* and replies with Shazam!.

The *expect_string* can be any part of the string expected from the remote string. Thus, the sample chat script could be written:

```
ogin: nuucp ssword Shazam!
```

and yield the same result.

Recommendation: You should omit the first letter of the login and password because some systems might use capital letters for one or both of the words and some may not. To avoid having to find out which way a system is and possible changes on the remote system, the first letters are omitted from the expect string.

The *expect_string* can be replaced with a string of the format:

```
expect_string-subsend_string-subexpect_string
```

where *subsend_string* and *subexpect_string* are text strings similar to *send_string* and *expect_string*. Hyphens separate the *expect_string*, the *subsend_string*, and the *subexpect_string*. With this format, your system waits for *expect_string* from the remote system and if it is received within a reasonable length of time, **uucico** responds with the *send_string* as described earlier. If it is not received, **uucico** sends the *subsend_string*, waits for the *subexpect_string*, and then finally sends the *send_string*.

For example, if you were using the chat script described earlier and there was noise on the line that garbled the login: string, the chat script would fail. However, the following chat script might work:

```
ogin:--login: nuucp ssword: Shazam!
```

This script waits for login: from the remote system. If it is not received, **uucico** replies by sending a null string (there is nothing between the two

hyphens) followed by a newline. **uucico** then again waits for `login:`. When it is received, `nuucp` is sent. The remainder of the script is identical to the earlier example.

If you would like an *expect_string* to wait a specific length of time for a match, you can suffix the *expect_string* with a tilde (~) followed by a number. The number is the number of seconds to wait for the *expect_string*. For example, the chat script

```
ogin:~10--login: nuucp ssword: Shazam!
```

waits 10 seconds for the string `login:` before continuing. You can use this suffix with *subexpect_string* as well.

Table 15 shows the escape sequences which you can use in a chat script.

Table 15. Escape characters usable in chat scripts

Escape	Description
""	Expect a null string
EOT	Send the end-of-transmission character
BREAK	Cause a BREAK
\b	Send a BACKSPACE
\c	Suppress newline or carriage return
\d	Delay for one second
\K	Send a BREAK
\n	Send a newline
\N	Send a NULL
\p	Pause for a fraction of a second
\r	Carriage return
\s	Send a space
\t	Send a tab
\\	Send a backslash
\ ~	Expect a tilde
\ddd	Send the EBCDIC character with octal code <i>ddd</i> . For example, use <code>\100</code> to represent a space character.

The Devices file

The Devices file contains information for direct links and network connections. Each entry has the format:

```
type dataport - speed dialer-arg ...
```

Only TCP/IP connections are supported. Specify this entry in the Devices file:

```
TCP - - -
```

The following list describes what each field represents:

type Describes the type of link. The following keyword is a valid entry for this field:

TCP A link through TCP/IP. When TCP is specified, each of the other fields should contain a hyphen. You must also specify TCP in the **dialer** field of the **Dialers** file.

dataport

Is the device name of the port used to make the connection. For TCP/IP, specify a – (hyphen).

– An obsolete field, *dialer-port*. Specify a hyphen.

speed This speed must match the value of the speed field in the corresponding **Systems** file entry. For TCP/IP, specify a – (hyphen).

dialer-arg

Contains a list of one or more pairs with the format

dialer arg

where *dialer* is the dialer name and *arg* is an argument to pass to that dialer. For TCP/IP, leave this blank.

The Dialers file

The **Dialers** file provides dialing instructions for the dialers referred to in the **Devices** file. Each entry has the following format:

```
dialer subs expect-send [expect-send ...]
```

Only TCP/IP is supported, so the entry that must appear in this file is:

TCP

The following list explains what each of these fields represents:

dialer The type of dialer. For TCP/IP, specify TCP.

subs Contains a string of characters. For TCP/IP, leave this blank.

expect_send

Is similar to the *chat_script* field in the **Systems** file. The major difference is the set of escape characters which can be used.

The Dialcodes File

The Dialcodes file contains abbreviations that can be included in the phone numbers specified in the **Systems** file. For TCP/IP, in this file you specify:

- -

The Permissions File

The **Permissions** file is used to control the access that remote systems have to data and programs on the local system. Specifically, it is used to specify:

- Which systems can establish a **uucico** connection
- The areas in the file system that a remote system can read or write from
- The commands that the remote system can run on the local system
- If the local system will process its waiting work when contacted by another system
- An alias for the local system
- A different public directory

The format of each entry in the **Permissions** file is:

```
LOGNAME=userid [MACHINE=system] option=value [option=value] ...
```

or

```
MACHINE=system [LOGNAME=userid] option=value [option=value] ...
```

where *option* is one of the options listed later in this section and *value* is one or more values that you want to set for that option. Options and values are case-sensitive. When specifying multiple values for an option, separate the values with a colon (:). Here is a sample entry:

```
MACHINE=ME READ=/ WRITE=/ COMMANDS=ALL
MACHINE=site1:site2:SITE3 \
    READ=/ \
    WRITE=/ \
    COMMANDS=uucp:cat:ls
LOGNAME=NUUCP \
    READ=/ \
    WRITE=/ \
    SENDFILES=yes \
    DEBUG=9 \
    VALIDATE=site1:site2:SITE3
```

The Permissions file can also contain blank lines (which are ignored) and comment lines. To indicate that a line is a comment line, use a number sign (#) as the first character in the line.

Each entry must contain the LOGNAME option or the MACHINE option, or both. Both options are used to identify an entry that applies to a remote system when it is processing its file transfer requests. The difference between them is based on which system initiates the connection:

- LOGNAME=userid entries apply to a remote system when it initiates the connection by logging onto your system as *userid*.
- MACHINE=system entries apply to a remote system when your system initiates the call to *system*.

If your system initiates the connection, your system first processes any queued file transfer requests that it has. When this is complete, the remote system can indicate that it has file transfer requests queued on its system that it would like to process. If the correct permissions are set, control switches to the remote system which then processes its file transfer requests. At this point, the MACHINE entry options are used for the remote system.

If your system does not need to differentiate Permissions options based on which system initiates the call, then LOGNAME and MACHINE can appear in the same entry.

These are the LOGNAME and MACHINE options:

LOGNAME

Indicates the user IDs that remote systems can use when logging on to your system. For z/OS systems, these names should be specified in uppercase unless USERIDALIASTABLE is used to define lowercase or mixed-case aliases. See “USERIDALIASTABLE” on page 37 for more information about defining user aliases.

MACHINE

Specified as MACHINE=system, this indicates the remote systems that your system can call using the other options specified in this entry. The system name specified here must also be specified as a system in the systems file. If you set this option to OTHER, the options specified apply to any remote system not specified by a MACHINE option in another entry. For remote z/OS systems, these names will usually be uppercase. Contact the remote system’s UUCP administrator to make sure that the names are uppercase.

Permissions for **uux** commands (which are executed by **uuxqt**) are based on MACHINE entries regardless of which system initiates the call.

Following is the list of valid options that are used with either LOGNAME or MACHINE entries, or with both. Options are marked with an (L) or an (M) to indicate that they are intended for LOGNAME or MACHINE entries or for both (L,M). An option used in an entry for which it is not intended will be ignored.

READ (L,M) Indicates which directories **uucico** can read. By default, this is the home directory of user uucp (**/usr/spool/uucppublic**). Remember that **uucico** runs with the effective UID of UUCP, so you must permit the *uucp* user or *uucpg* group to read from these directories.

WRITE

(L,M) Indicates which directories **uucico** can write to. By default, this is **/usr/spool/uucppublic**, the home directory of user uucp. Remember that **uucico** runs with the effective UID of UUCP, so you must permit the *uucp* user or *uucpg* group to write to these directories.

NOREAD

(L,M) Indicates that files in the specified directories cannot be read. If a directory is specified by both READ and NOREAD, files in that directory cannot be read. The public directory can always be read (even if specified on NOREAD).

NOWRITE

(L,M) Indicates that files in the specified directories cannot be written to. If a directory is specified by both WRITE and NOWRITE, files in that directory cannot be written to. The public directory can always be written to (even if specified on NOWRITE).

PUBDIR

(L,M) Indicates the public directory. By default, this is the home directory of user **uucp** (**/usr/spool/uucppublic**).

If you are going to change PUBDIR on your system, you need to have an additional MACHINE entry for your local site. See “Add an entry to the permissions file” on page 226. Consider this example:

```
uucp remote_site!/file1 local_site!~/file1
```

When **uucp** processes this command it looks for a MACHINE=local_site entry to find the value for PUBDIR.

DEBUG

(L,M) Indicates the verbosity of the debugging information. Set this to a number between 0 and 9. Level 0 provides terse debug messages while level 9 provides verbose output. This output will be stored in **/usr/spool/uucp/LOGFILE** to aid you in debugging communications problems when remote systems call you.

REQUEST

(L,M) Indicates whether requests made by remote systems to transfer data *from* your system are allowed. This option can be used to protect data on your system from being read by remote systems.

- If set to yes, remote systems can read data from those directories it is authorized to read from.
- If set to no, a remote system can write data to your system, but cannot read data irrespective of the value of the READ option. This is the default.

This option only applies to requests originating from the remote system. This option has no effect on file transfer requests that originate on your system.

SENDFILES

(L) Indicates if your system will process its own queued file transfer requests after the remote system has initiated the connection and completed its file transfer requests. The SENDFILES option allows the local system to control when its queued file transfer requests are processed.

- If this option is set to **yes**, your system will process its queued requests after the remote system has completed processing its own.
- If this option is set to **call**, your system will only process its own file transfer requests when it initiates the connection with the remote system. This is the default.

VALIDATE

(L) Names the remote systems that can login to your system using the user IDs given by LOGNAME. If another system attempts to login using this user ID, **uucico** refuses the connection.

COMMANDS

(M) Indicates the commands that the remote system can execute on your system.

By default, the **uucp** command is not permitted, which means that by default your local system is a terminal, or leaf-node, connection. To allow a remote system to transfer files through your local system, specify **uucp** for the COMMANDS option.

- To specify more than one command, separate the command names with a colon (:). For example, **COMMANDS=uucp:ls**.
- To prohibit all commands, do not use the COMMANDS option.
- To allow access to all commands, set this option to **ALL**.

MYNAME

(M) Tells the remote system that the name of your local system is the specified value rather than the name given by **uname -n**.

Entries in the permissions file: An example may help to explain how the entries in the **Permissions** file work.

Tip: Whenever a z/OS system or uucp login is specified, the name should be specified in uppercase

Suppose that the system named North in the sample network has the following **Permissions** file.

```
LOGNAME=uwest MACHINE=west READ=/ WRITE=/ \
COMMANDS=uucp:mail NOREAD=/usr/private \
NOWRITE=/usr/private SENDFILES=yes REQUEST=yes \
VALIDATE=west

LOGNAME=nuucp MACHINE=OTHER REQUEST=yes \
SENDFILES=call
```

The first entry in this file specifies the options that are in effect when a remote system logs in as *uwest*. Because of the **VALIDATE=west** option, the only remote system that can use this user ID is West. When West calls North and logs in as *uwest*, it can read from and write to all directories except the ones starting with */usr/private* and can execute the commands **uucp** and **mail** on North's system. This entry also includes the **MACHINE=west** option, meaning the options given also apply when North has called West and control has been transferred to north's **uucico** utility. Because **REQUEST=yes** and **SENDFILES=yes**, either system can request or send working files.

The second entry specifies the options in effect when a remote system logs in with the NUUCP user ID. Because MACHINE=OTHER, these options will also apply when north has called any remote system except west (which has its own entry) and control has been transferred to north's **uucico**. Files can only be read from or written to the **/usr/spool/uucppublic** directory (no READ or WRITE options to change the default). Either system can request files from the other, but working files will only be transferred from north when it calls the remote system.

How uucico uses configuration files

When **uucico** is invoked, it searches the information provided by the **Systems** file (and compiled into the **config** file) for the remote system indicated on its command line. If the sched field of the matching entry indicates that it is valid to contact the remote system at that time, **uucico** then checks to see if a connection with the remote system is already in progress (a lock exists in **/usr/spool/locks** for the system or system IP address). If a connection is already in progress, **uucico** will not initiate another connection at this time. Otherwise, **uucico** will attempt to open a TCP/IP connection using the remote system's IP address. If successful, it uses the contents of the chat_script field to complete the connection.

Compile the configuration files

In z/OS UNIX, UUCP does not use the configuration files directly. Instead, it uses a special configuration file named **config** which is created when the administrator runs the **uucc** utility to compile the configuration files.

After you have set up the configuration files, change directories and then run the commands:

```
cd /usr/lib/uucp
uucc
```

This creates a compiled **config** file in **/usr/lib/uucp** which contains all the information specified in the individual configuration files. If a previous **config** file existed, it will be renamed to **config.old** before the new one is created.

Rule: The config file must be owned by the uucp user ID. If you run **uucp** from any other user ID, you must change the owner of the config file from that user ID to uucp.

Do not edit the **config** file directly. If you need to make changes to your configuration, first edit the configuration files and then run **uucc** again.

Create working directories for the local and remote systems

UUCP requires a working directory in **/usr/spool/uucp** for the local system and for each system defined in the Systems file. Each directory must be owned by uucp and have **uucpg** as its group ID. If you create the directories with the **uucp** user ID, this will happen automatically. Otherwise, you will need to chown these directories from a superuser user ID.

- Create a working directory for the local system. Enter:

```
mkdir -m 774 /usr/spool/uucp/${uname -l}
```

(\${uname -l}) will be replaced with the name of your system). If the directory is not owned by uucp and uucpg, enter:

```
chown uucp:uucpg /usr/spool/uucp/${uname -l}
```


- Create working directories for remote systems. (If you are setting up your uucp environment for the first time, see the “Tip” below.) For each remote system, enter:

```
mkdir -m 774 /usr/spool/uucp/system
```

where *system* is the name of the remote system.

If the directories are not owned by uucp and uucpg, enter:

```
chown uucp:uucpg /usr/spool/uucp/system
```

where *system* is the name of the remote system.

Tip: If you are setting up your UUCP environment for the first time, you can create working directories for all remote systems listed in the Systems file by executing the following commands:

```
cd /usr/spool/uucp
mkdir -m 774 $(uname)
chown uucp:uucpg $(uname) # if necessary.
```

\$(uname) will be replaced with a list of all systems defined in the Systems file

Schedule periodic UUCP transfers with cron

UUCP provides two daemons (**uucico** and **uuxqt**) which establish communication sessions, transfer data, and execute commands according to the requests scheduled by **uucp** (for file exchange) and **uux** (for command execution). **uux** will invoke **uucico** unless the command is a local one, in which case it will invoke **uuxqt** to process the local command immediately. While you can invoke these daemons interactively as the need arises, this becomes inconvenient if many users become dependent on UUCP's capabilities, or if the system must receive data from other UUCP systems according to some regular schedule.

You may need to use **cron** for two reasons:

1. To process requests that were left on the request queue when **uucico** could not connect.
2. To get files that are waiting to be received from other systems.

The **cron** facility can be used to run the UUCP daemons according to a fixed schedule such as:

- Monday through Friday at 7:30 p.m.
- Each day at 8:00 a.m. and noon
- Every 15 minutes starting at midnight

It is also used to initiate work on behalf of others at predefined times. A **crontab** file defines the work to be done for a user and the schedule for running it. Use the **crontab** command to create the **crontab** file. After a user creates a **crontab** file (assuming that the **cron** facility has been configured by the administrator), **cron** initiates the work according the schedule specified.

If the UUCP daemons are running from **cron** and encounter an error, they send mail to the user who ran the **uucp** or **uux** command that the daemons are processing. The daemons log their status and errors in two files: **/usr/spool/uucp/ERRLOG** is used to log errors and **/usr/spool/uucp/LOGFILE** is used to log non-error status. Check those files if the daemons **uucico** or **uuxqt** do not seem to be running correctly.

Creating a crontab entry

You can use **cron** to run the UUCP daemons on a fixed schedule. For example, if you want to run the UUCP daemons every Monday through Friday at 7:30 p.m., you would:

1. Log on as UUCP or **su** to the UUCP user ID
2. Enter the following **echo** command to create a working copy of the desired crontab entry:

```
echo '30 19 * * 1-5 /usr/lib/uucp/uucico; /usr/lib/uucp/uuxqt;' >tfile
```

where

0 means zero minutes

19:30 means 7:30 p.m.

* means no selected day of the month

* means no selected month of the year

1-5 means Monday through Friday

/usr/lib/uucp/uucico; /usr/lib/uucp/uuxqt; are the commands to be run

>tfile directs the output to a temporary file

3. Enter the **crontab** command to activate your request.

```
crontab tfile
```

4. To display your current crontab entries, enter the following command:

```
crontab -l
```

Tip: Do not issue the **crontab** command without any options. If you do, the system will erase your current crontab entries and accept new crontab entries from the terminal. If you accidentally enter **crontab** without any options, end it with the INTERRUPT key, which by default is <Ctrl-C>.

Example of schedules

Here are some examples of other schedules and their crontab entries:

1. Every day at 8:00 a.m. and noon:

```
0 8,12 * * * /usr/lib/uucp/uucico; /usr/lib/uucp/uuxqt;
```

where:

0 specifies what minute

8 and 12 specify 8 a.m. and noon

* is every day of the month

* is every month of the year

* is every day of the week

2. Every fifteen minutes starting at midnight:

```
0,15,30,45 * * * * /usr/lib/uucp/uucico; /usr/lib/uucp/uuxqt;
```

where

0,15,30,45 indicates every fifteen minutes

* specifies every hour of the day

* specifies every day of the month

* specifies every month of the year

* specifies every day of the week

There are many other scheduling possibilities. For more information, see the **crontab** command in *z/OS UNIX System Services Command Reference*.

Controlling calls to each system

By default, **uucico** attempts connection to every system listed in the Systems file.

To reduce these attempts, you can code acceptable “call out” times for each system

in the **Systems** file. In addition, each system will be contacted even if no data transfers or remote command executions have been requested on the local system. (This is to receive data transfers or local command executions that were requested on the remote system.)

You can specify different **uucico** crontab entries for different systems. Each of these crontab entries will specify a command of the form

```
uucico -r1 -s site
```

where *site* is the name of the remote site to be called.

This sample **crontab** entry calls the system named North every hour on the hour:

```
0 * * * * /usr/lib/uucp/uucico -r1 -s north
```

This sample **crontab** entry calls the system named East at 12:00 noon and 7:00 p.m. each day from Monday to Friday:

```
0 12,19 * * 1-5 /usr/lib/uucp/uucico -r1 -s east
```

Testing the connection

Once your local system is configured and a remote system is configured, you must test the connection.

1. Change directories to the public UUCP directory.

```
cd /usr/spool/uucppublic
```

2. Queue up a file request with this command:

```
uucp -mr testfile remote!~/
```

where *testfile* is the name of a file in the public UUCP directory and *remote* is the name of the remote system.

3. Force a connection:

- a. If the remote system calls your system, have the remote system administrator attempt a connection.
- b. If your system calls the remote system, force a connection with this command:

```
/usr/lib/uucp/uucico -f -r 1 -s remote -x 5
```

where *remote* is the name of the remote system.

4. If everything has been configured correctly, the file is transferred and you can read the mail with **mailx**.

Problem determination for connection problems

If there are problems, check the configuration for this connection. In solving UUCP problems, try to determine how far the connection proceeds before failing. Every UUCP connection goes through these stages:

1. One system establishes a TCP/IP connection with another.
2. The contacted system sends a login prompt and the calling system logs in.
3. The systems negotiate protocols.
4. Files are exchanged.
5. The calling system hangs up.

Try to determine at what stage your connection breaks down. Examine the file **/usr/spool/uucp/LOGFILE** for clues.

Contacting the remote site

If **uucp** cannot establish a connection with the remote system, one of the IP address for the remote system may be wrong or the network path between your system and the remote system may be down. You should be able to ping the remote system to confirm this. For example, if the address of the remote system west is west.ibm.com, enter:

```
tso 'ping west.ibm.com'
```

The following indicates a successful connection:

```
PING V3R1: PINGING HOST west.ibm.com (120.40.41.3).  
USE ATTN TO INTERRUPT.  
PING: PING #1 RESPONSE TOOK 0.043 SECONDS. SUCCESSES SO FAR 1.
```

Calling system login

If the call is made but is not answered, then the login sequence may be at fault. Check **/usr/spool/uucp/LOGFILE** for the record of the sequence exchanged.

- Check the login send/expect sequences specified in the Systems file.
- If the log shows failed logins and the message “you are unknown to me,” confirm that both systems have the correct login name and password and their configurations are set up correctly.

Maintaining UUCP

To maintain UUCP, you need to:

- Read and remove log files periodically—check **/usr/spool/uucp/LOGFILE**.
- Use the **uustat** command to check the status file to ensure that files are transferring to remote systems. Periodically update the configuration files to reflect changes in your system.

Cleaning up UUCP files

Some UUCP files will reside on your UUCP system after it is configured. Here are some pointers on how to clean up old files and make sure that all necessary files are present.

The spool directory

The spool directory holds all work requests and all log files for UUCP. File transfers can be requested by the **uucp** and **uux**.

For each remote system specified in the Systems configuration file, there is a subdirectory in the **/usr/spool/uucp** directory named for the system (for example, the subdirectory for a remote system named South is **/usr/spool/uucp/south**). This subdirectory contains:

- File transfer requests for a remote system.
- Data files for file transfer requests for a remote system.

UUCP data files are created from:

- **uucp** with the **-C** option.
- **uux** requests whose arguments are files that are not on the system running the requested command.
- Execute files, commands that a user on the remote system has requested be run on your system. When you run **uuxqt**, it looks for files in those directories and runs the commands indicated (if all of the permissions are correct).

- The **.Xqtdir** subdirectory, which acts as a working directory when **uuxqt** runs remote commands. After finishing a command, **uuxqt** removes working files from this directory.

Log files, lock files, status files, and working files

UUCP creates system files in the spool directory or in its *system* subdirectories. The following types of system files are created:

Log files

Records of events such as file transfers, deletions, attempts to connect with other systems, and system errors. The spool directory contains the following log file: **/usr/spool/uucp/LOGFILE**. It contains the record of when jobs were queued and executed.

Attention: This log file can grow indefinitely. You should edit or delete it on a regular basis.

Lock files

Temporary files created to prevent two programs writing to a file or device simultaneously. These files will have a name of **LCK..site** and **LCK..site_address**.

Status files

Records of the most recent unsuccessful attempt to contact a remote system. There is one status file for each remote system you contact; status files are named **/usr/spool/uucp/.Status/site**, where *site* is the remote site's name.

You can use the **uustat -q** command to view the contents of the status file.

A status file is only created if the last attempt to contact a system was unsuccessful. The status file is not required for UUCP to attempt another call.

Working files

Command, data, and execute files for the UUCP file transfer programs, stored in the appropriate subdirectory for the system. These files are described in an appendix in *z/OS UNIX System Services Command Reference*.

Displaying information about recorded UUCP events

You can use the **uulog** command to display information about recorded UUCP events, such as file transfers and remote command execution.

Notifying remote systems about password changes

When you change the password of a user ID (for example, NUUCP) that is used for a remote system to login to the local system, you must notify each remote system to update its Systems file chat script with the new password.

Chapter 13. Using Enhanced ASCII functionality

Enhanced ASCII functionality makes it easier to port internationalized applications developed on ASCII platforms, or for them, to z/OS platforms by providing conversion from ASCII to EBCDIC, and from EBCDIC to ASCII. This chapter discusses Enhanced ASCII and explains how to set it up.

Overview

This section provides information that will help you decide whether to implement Enhanced ASCII.

To begin with, z/OS is an EBCDIC platform. It has devices that are configured for EBCDIC; it also has programs that are compiled to handle the EBCDIC encoding of characters. If you implement Enhanced ASCII, the basic EBCDIC nature of a z/OS platform remains. For example, the z/OS shell and utilities continue to be EBCDIC programs. However, if C programs have been compiled as ASCII, the EBCDIC nature can be partially hidden.

C programs that have been compiled as ASCII use ASCII locales. These ASCII locales are produced by using the **-A** option of **localedef**.

Before Enhanced ASCII, you could use **iconv** to convert files from ASCII to EBCDIC. Enhanced ASCII introduces automatic conversion which, in some cases, is an alternative to **iconv**. *z/OS UNIX System Services Porting Guide* contains examples of automatic conversion.

Enhanced ASCII also provides support for file tagging. File tags are a way to identify the code set of text data within files. File tags are used during automatic conversion.

In this chapter

This chapter covers the following subtask.

Subtask	Associated procedure (see . . .)
Setting up enhanced ASCII	"Steps for setting up Enhanced ASCII" on page 246

Limitations of Enhanced ASCII

This section explains under what conditions you can use Enhanced ASCII.

- A subset of C headers and functions is provided in ASCII. For more information, see *z/OS C/C++ Run-Time Library Reference*.
- The only way to get to the ASCII version of functions and the external variables **environ** and **tzname** is to use the appropriate IBM header files.
- ASCII applications may read, but not update, environment variables using the **environ** external variable. Updates to the environment variables using **environ** in an ASCII application causes unpredictable results and may result in an abend. Language Environment maintains two equivalent arrays of environment variables when running an ASCII application, one with EBCDIC encodings and the other with ASCII encodings. All ASCII compile units that use the **environ** external variable must include `<stdlib.h>` so that **environ** can be mapped to access the

ASCII encoded environment strings. If `<stdlib.h>` is not included, **environ** will refer to the EBCDIC representation of the environment variable strings.

Enhanced ASCII provides limited conversion of ASCII to EBCDIC, and EBCDIC to ASCII. The character set or alphabet that is associated with any locale consists of the following:

- A common, XPG4-defined subset of characters such as POSIX portable characters
- A unique, locale-specific subset of characters such as NLS characters

The conversion only applies to the portable subset of characters that are associated with a locale. Only the EBCDIC IBM-1047 encoding of portable characters is supported.

You might encounter unexpected results in the following situations:

- If Enhanced ASCII applications run in locales that contain non-Latin Alphabet Number 1 NLS characters, C-RTL functions might copy some of the locale's non-Latin 1 NLS characters into buffers that the application is writing to stdout or other HFS files. The non-Latin Alphabet Number 1 NLS characters would then cause problems during automatic conversion.
- Language Environment applications might select non-English message files. If the NATLANG run-time option is not ENU or UEN, then conversion does not take place. The messages are presented to the file system write routine in EBCDIC, before any automatic conversion takes place. If the automatic conversion is to EBCDIC, then there will be a problem because EBCDIC cannot be converted to EBCDIC.

Setting up Enhanced ASCII

Recommendation: You should limit the enabling of automatic conversion to the smallest environment possible. One way to accomplish this is by using the `_BPXK_AUTOCVT` environment variable, optionally with the `FILETAG` run-time option.

It is important to understand that file tagging and enabling automatic conversion are independent operations. You can tag files without enabling automatic conversion, and vice versa. Because of this, it is possible to have many tagged files without any conversion occurring. However, if you enable automatic conversion for the entire system by using the `AUTOCVT` statement in `BPXPRMxx`, every tagged file becomes subject to conversion by any program that reads from or writes to those tagged files. Thus, programs may be processing converted data even though they do not support it. (An example would be an EBCDIC program that expects to read an ASCII file as ASCII data.) For those reasons, it is a good idea to limit the enabling of automatic conversion to the smallest environment possible using `_BPXK_AUTOCVT` and, if applicable for the C run time environment, the `FILETAG` run-time option.

Another reason to limit the scope of automatic conversion is that `AUTOCVT(ON)` incurs a slight performance penalty, even for programs that do not automatically convert data. For this reason, you may want to limit the scope of automatic conversion.

Steps for setting up Enhanced ASCII

Before you begin: You need to have an overall understanding of the limitations of Enhanced ASCII, as explained in “Limitations of Enhanced ASCII” on page 245.

Perform the following steps to set up Enhanced ASCII.

1. Set up Enhanced ASCII. Base your choice on your particular situation.

If this situation exists . . .	Then use
The application is written in C/C++.	The FILETAG run-time option with the _BPXK_AUTOCVT environment variable.
The application is run in the z/OS UNIX shell or BPXBATCH.	The _BPXK_AUTOCVT environment variable.
You are enabling automatic conversion for the z/OS UNIX environment.	AUTOCVT(ON) statement in the BPXPRMxx parmlib member. Tip: You can use the SETOMVS and SETOMVS operator commands to turn AUTOCVT on or off. See <i>z/OS MVS Initialization and Tuning Reference</i> for more information about those commands.

2. Assign the appropriate file tag for each file that is to be converted. Base your choice on your particular situation.

If you choose this method . . .	Then this happens . . .
Issuing the chtag command.	Files are permanently tagged.
Mounting a file system with the TAG parameter.	Files are temporarily tagged. All untagged files in the file system that is being mounted are implicitly tagged. When the file system is unmounted, the tags are lost.
Issuing the F_SETTAG subcommand of the BPX1FCT (fcntl) callable service from a program.	Files are either temporarily or permanently tagged, depending on the input parameters. See <i>z/OS UNIX System Services Programming: Assembler Callable Services Reference</i> and <i>z/OS C/C++ Run-Time Library Reference</i> for more details.
Issuing BPX1CHR (chattr) callable service from a program.	Files are permanently tagged. See <i>z/OS UNIX System Services Programming: Assembler Callable Services Reference</i> and <i>z/OS C/C++ Run-Time Library Reference</i> for more details.
Issuing fopen() or popen() with the 'text' option, and using the C-RTL FILETAG(,AUTOTAG) run-time option.	New or empty files are automatically tagged at first write(). Programs that use this form of opening a file are already set up for tagging, and require the least effort to set up automatic conversion.

3. Assign a coded character set identifier (CCSID) to each program or thread in the shell. By default, the initial CCSID for every thread is 1047 (EBCDIC).

Examples:

- For entire programs written in C/C++, use the ASCII compiler to change it to 819 (ISO8859-1 ASCII).
- For C/C++ threads, use the F_CONTROL_CVT subcommand of fcntl().
- For Assembler programs and threads, use the F_CONTROL_CVT subcommand of the BPX1FCT callable service. F_CONTROL_CVT sets the CCSID of the program associated with each opened file. (That is, the program CCSID can be different depending on which file is chosen.)

- Set field Thliccsid using the mapping macro BPXYTHLI.

When you are done, you have set up Enhanced ASCII.

Chapter 14. Managing operations

Overview

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

In this chapter

This chapter covers the following subtasks.

Subtasks	Associated procedure (see . . .)
Ending a specified process	"Steps for ending a specified process" on page 250
Shutting down z/OS UNIX	"Steps for shutting down z/OS UNIX using F BPXOINIT,SHUTDOWN=..." on page 252 "Steps for shutting down z/OS UNIX using F OMVS,SHUTDOWN" on page 257
Doing partial shutdowns for JES maintenance	"Steps for partial shutdowns for JES2 maintenance" on page 254
Dynamically adding FILESYSTYPE parameters in BPXPRMxx	"Steps for activating the HFS file system for the first time" on page 261 "Steps for activating a single sockets file system for the first time" on page 262 "Steps for activating a multiple socket file system for the first time with Common INET" on page 263 "Steps for increasing the MAXSOCKETS value" on page 263 "Steps for adding another sockets file system to an existing Common INET (CINET) configuration" on page 264

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

- "Comparing UNIX security and z/OS UNIX security" on page 290.
- "Establishing the correct level of security for daemons" on page 291.

See *z/OS C/C++ Programming Guide* for more information about writing a daemon program.

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

Ending processes

This section explains how to end a specified process.

Steps for ending a specified process

Before you begin: You need to know which processes you want to end and whether they are active. Use the DISPLAY OMVS operator command or the **ps** command to display all active processes.

Examples:

1. To display process information for all z/OS UNIX address spaces, issue:
`DISPLAY OMVS,A=ALL`
2. To display a particular ASID, where *asid* is a specified ASID:
`DISPLAY OMVS,A=asid`
3. To display information on all accessible processes, providing that you have the appropriate privileges:
`ps -elf`

Perform the following steps to end a specified process.

1. Send a SIGTERM via the shell **kill** command or a TERM via the MODIFY console command.

Examples:

- a. To send a SIGTERM via the shell **kill** command:

```
kill -s term pppp
```

where *pppp* is the process identifier.

- b. To send a TERM via the MODIFY console command:

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

where *pppp* is the process identifier.

If the process is not ended, then go to Step 2. Otherwise, you have canceled the process and you are all done.

-
2. Send a SIGKILL via the shell **kill** command or a FORCE via the MODIFY console command.

Examples:

- a. To send a SIGKILL via the shell **kill** command:

```
kill -s kill pppp
```

where *pppp* is the process identifier.

- b. To send a FORCE via the MODIFY console command:

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

where *pppp* is the process identifier.

If the process is still not ended, then go to Step 3. Otherwise, you have canceled the process and you are all done.

-
3. If the previous steps did not end the process, then you can use the CANCEL command. Issue:

```
CANCEL jobname,a=asid
```

Example: For a user with a TSO/E user ID of JOE, Figure 49 shows how to obtain the ASIDs for the user's work and then cancel the user's process that is running the **sleep 6000** command.

```
display omvs,u=joe
BPX0001I 17.12.23 DISPLAY OMVS 361
```

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

```
cancel joe3,a=1b
```

Figure 49. Console display for a CANCEL command

When you are done, you have ended the specified process.

Ending threads

An operator can end a thread without disrupting the entire process.

Ending threads with the MODIFY command

The syntax of the MODIFY command to end a thread is:

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

where

- **pid** indicates the process identifier (PID) of the thread to be ended. The PID is specified in decimal form as displayed by the D OMVS command.
- **tid** indicates the thread identifier (TID) of the thread to be ended. The TID is 16 hexadecimal (0-9,A-F) characters as displayed by the following command:

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

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

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

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

Guideline: If a thread in a process is hung, the operator can use a MODIFY console commands to terminate the thread without ending the entire process. Use the TERM keyword first. If that does not succeed, then use FORCE.

- To allow the signal interface routine to receive control before the thread is ended:

F BPXOINIT,TERM=pppppppp.tttttttttttttt

where pppppppp is the process identifier and tttttttttttttt is the thread identifier.

- To terminate the thread without allowing the signal interface routine to receive control:

F BPXOINIT,FORCE=pppppppp.tttttttttttttt

where pppppppp is the process identifier and tttttttttttttt is the thread identifier.

Planned shutdowns using F BPXOINIT,SHUTDOWN=...

When you are doing a planned shutdown of z/OS UNIX, follow the instructions in “Steps for shutting down z/OS UNIX using F BPXOINIT,SHUTDOWN=...” As part of a planned shutdown, you need to prepare the file systems before shutting down z/OS UNIX by issuing either F BPXOINIT,SHUTDOWN=FILEOWNER or F BPXOINIT,SHUTDOWN=FILESYS. This synchronizes data to the file systems and possibly unmounts or moves ownership of the file systems. When SHUTDOWN=FILEOWNER is used, the system is also disabled as a future file system owner via move or recovery operations until z/OS UNIX has been recycled.

The steps for shutting down z/OS UNIX file systems are the same whether or not the system is participating in shared HFS. However, in a shared HFS environment, the resulting system actions are more complex, because they may involve the movement of file system ownership between systems in the shared HFS group. For more information about the system actions that may occur in a shared HFS environment, see “Shared HFS implications during a planned shutdown of z/OS UNIX” on page 180.

To shut down the system as part of JES maintenance without re-IPLing the system, see “Partial shutdowns (for JES2 maintenance)” on page 254.

Steps for shutting down z/OS UNIX using F BPXOINIT,SHUTDOWN=...

Before you begin: You need to notify users that the system is being shut down and ask them to log off. If you do not shut down and quiesce the UNIX workload, these critical system functions might be ended abnormally during the shutdown, which may cause several failures on the system. As a result, the system might not be shut down successfully.

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

Example:

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

2. Use the **wall** command to send a similar note about the impending shutdown to all shell users who are logged on.

Example:

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

Perform the following steps to shut down z/OS UNIX using F BPXOINIT,SHUTDOWN.

1. Prevent new TSO/E logons and shut down other z/OS subsystems (such as CICS® and IMS™), following your usual procedures.

2. Shut down all JES initiators.

3. Unmount all NFS-mounted file systems as part of the normal shutdown process.

4. Use normal shutdown procedures to end all file system address spaces such as TCP/IP and DFSS. Do this after the final warning has been sent to users that the system is terminating.

5. Terminate running daemons such as **inetd**. Then terminate any remaining processes.

Tips:

- To get a list of daemons that are running, issue:

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

OMVSKERN is the user ID that is used for the kernel and daemons.

- To display all processes (most daemons will have recognizable names), issue:

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

Then use the F BPX0INIT,TERM=xxxxxxx operator command or the **kill** command to terminate those processes.

6. Move or unmount all file systems (including the root file system). Issue::

```
F BPX0INIT,SHUTDOWN=FILEOWNER
```

or

```
F BPX0INIT,SHUTDOWN=FILESYS
```

7. Take down JES. At this point, there may still be a number of initiators that are provided by WLM for use on fork and spawn. These initiators time out after 30 minutes on their own, but you can end them by issuing:

```
F BPX0INIT,SHUTDOWN=FORKINIT
```

When you are done, you have ended all of the processes. You can do any of the following:

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

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

Partial shutdowns (for JES2 maintenance)

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

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

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

Steps for partial shutdowns for JES2 maintenance

Before you begin: You need to notify users that the system is being shut down and ask them to log off. If you do not shut down and quiesce the UNIX workload, these critical system functions might be ended abnormally during the shutdown, which may cause several failures on the system; as a result, the system might not be shut down successfully.

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

Example:

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

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

Example:

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

Perform the following steps to accomplish a partial shutdown.

1. Shut down z/OS UNIX.

```
F BPX0INIT,SHUTDOWN=FORKS
```

Result: You have ended all forked and non-local spawned address spaces on the system. If the operator receives a success message, the shutdown can be continued.

Tip: A failure message means that some forked processes or non-local spawned address spaces could not be ended. To find these processes, issue:

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

To terminate them, issue:

```
F BPX0INIT,FORCE,FORCE=xxxxxxxx
```

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

-
2. Stop the file system colonies that were started under JES (those without SUB=MSTR specified when they were defined). Use normal shutdown

procedures to close all file system address spaces such as Network File System Client (NFSC) and the Distributed File System Cache Manager (DFSCM).

For NFSC, determine the name of the process that was used to start this colony and then cancel the process.

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

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

When you are done, you have partially shut down z/OS UNIX. New fork and spawn activity cannot be done; however, it is still possible for batch jobs and TSO users to use z/OS UNIX services. Now you can do whatever corrective or maintenance actions that were needed for JES2, such as restarting it.

Tip: To restart z/OS UNIX:

1. Issue the MODIFY (F) command.

```
F BPX0INIT,RESTART=FORKS
```

2. Restart the file system address spaces.

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

For DFSCM, respond to the operator message BPXF014D.

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

Result: You have restarted z/OS UNIX.

File system shutdown

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

- F BPX0INIT,SHUTDOWN=FILEOWNER
- F BPX0INIT,SHUTDOWN=FILESYS

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

The steps for shutting down z/OS UNIX file systems are the same whether or not the system is participating in shared HFS. However, in a shared HFS environment, the resulting system actions are more complex, because they may involve the movement of file system ownership between systems in the shared HFS group. For more information about the system actions that may occur in a shared HFS environment, see “Shared HFS implications during a planned shutdown of z/OS UNIX” on page 180.

Planned shutdowns using F OMVS,SHUTDOWN

“Steps for shutting down z/OS UNIX using F OMVS,SHUTDOWN” on page 257 describes how to use the F OMVS,SHUTDOWN operator command for a planned shutdown and re-IPL. Consider using it if you plan to recustomize and reinitialize the z/OS UNIX environment without re-IPLing. Use of F OMVS,SHUTDOWN along with F OMVS,RESTART may allow you to avoid a system outage by providing the ability to shut down and then reinitialize the z/OS UNIX environment without the need for a re-IPL.

Guideline: Sometimes F OMVS, SHUTDOWN cannot shut down z/OS UNIX completely and you have to do a re-IPL in order to correct the condition that is requiring the shutdown. With F OMVS,SHUTDOWN and F OMVS,RESTART, some reconfiguration tasks can be accomplished that otherwise would have required a re-IPL. These tasks include the following:

- Reconfiguring a system to go from a non-shared HFS system to a shared HFS system.
- Completely applying a new HFS file structure.

However, there are some tasks that you cannot accomplish using these commands, as follows:

- Installing maintenance to the z/OS UNIX component.
- Resolving severe system outages.

Follow the recommended pre-shutdown procedure described in “Steps for shutting down z/OS UNIX using F OMVS,SHUTDOWN” on page 257 when using F,OMVS,SHUTDOWN. If you do not, the risk of having to do a re-IPL is far greater.

If you want to shut down the system as part of JES2 maintenance and do not want to re-IPL the system, issue F BPXOINIT,SHUTDOWN=FORKS as described in “Partial shutdowns (for JES2 maintenance)” on page 254.

What F OMVS,SHUTDOWN does

Use the F OMVS,SHUTDOWN operator command when you want to do a planned shutdown, and may or may not be re-IPLing the system. You will be shutting down the entire z/OS UNIX system and all processes.

Only eligible running processes are shut down. Some processes may not be shut down because they have registered as a permanent process. Additionally, 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 ended.

Successful shutdowns

The shutdown succeeds only if all non-permanent z/OS UNIX processes end, all permanent processes are successfully checkpointed, and if all physical file systems are successfully quiesced. Otherwise, the shutdown request will fail. Nonpermanent processes within jobs that are not cleaned up cause the shutdown request to fail. These jobs are identified in messages so that you can force these jobs to end. At this point, because most z/OS UNIX processes have been ended, you should force the hung jobs to end and then try the shutdown again. Because some jobs may have ended abnormally, JES spool resources may have accumulated for these jobs; you will have to purge them using commands such as \$POJOBQ,READY.

At each phase of shutdown, it is possible that there could be a stall where no shutdown activity is occurring. That situation could cause the shutdown to hang. If such a situation is detected, the shutdown will wait approximately six minutes for the stall to resolve itself. If the stall does not resolve itself by then, the shutdown request will fail.

Because some z/OS UNIX resources are tied to components outside of the scope of the kernel (shared memory, mmap, shared libraries, for example), you must end any application that is using any of these resources before z/OS UNIX can be ended, including applications that are registered as permanent.

Tip: Because the F OMVS,SHUTDOWN support encompasses the existing support in the F BPXOINIT,SHUTDOWN= command, you do not need to issue F BPXOINIT,SHUTDOWN before using F OMVS,SHUTDOWN. If F OMVS,SHUTDOWN fails, z/OS UNIX services are reenabled whether or not a F BPXOINIT,SHUTDOWN= was done prior to the F OMVS,SHUTDOWN command. An F BPXOINIT command of any kind issued when OMVS is shut down is ignored.

Recommendation: Use F OMVS,SHUTDOWN 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.) An example of a system-wide resource that can be consumed due to the shutdown are non-reusable ASIDs. If colony address spaces are being used, a non-reusable ASID will be consumed for each colony address space that is shut down. For this reason, installations should plan on increasing the value set for the RSVNONR= parameter in the IEASYSxx parmlib member to account for the consumption of non-reusable ASIDs due to each shut down of OMVS. If this value is not increased, the installation may receive an error message after shutting down the system multiple times.

Steps for shutting down z/OS UNIX using F OMVS,SHUTDOWN

Before you begin: You need to notify users that the system is being shut down and ask them to log off. If you do not shut down and quiesce the UNIX workload, these critical system functions might be ended abnormally during the shutdown, which may cause several failures on the system. As a result, the system might not be shut down successfully.

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

Example:

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

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

Example:

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

1. Prevent new TSO/E logons.

2. 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.
Tip: You can use the D OMVS,A=ALL operator command to determine which applications, if any, require quiescing.

3. Unmount all remotely mounted file systems such as those managed by NFS. Doing so will prevent these file systems from losing data.

4. Terminate all file system address spaces such as TCP/IP and DFSS, using their recommended shutdown methods. If you do not shut them down before issuing F OMVS,SHUTDOWN, these system functions may terminate abnormally when the shutdown takes place. Existing colony PFS address spaces will be shut down. This may include NFS, for example.

Result: Now you can issue F OMVS,SHUTDOWN.

Notes:

1. After an F OMVS,SHUTDOWN request is accepted, jobs that attempt to use z/OS UNIX services for the first time will be delayed until the system is restarted. Terminating signals are sent to jobs that are already connected; these jobs will be ended abruptly.
2. After F OMVS,SHUTDOWN has completed, you can shut down the system completely via an IPL or by powering off.
Tip: You can completely restart and reinitialize the z/OS UNIX environment by issuing F OMVS,RESTART. You can also use it to change the configuration of z/OS UNIX services by specifying a different set of BPXPRMxx parmlib members when z/OS UNIX is started. For more information about F OMVS,RESTART, see *z/OS MVS System Commands*.
3. The steps for shutting down z/OS UNIX are the same whether or not the system is participating in shared HFS. However, in a shared HFS environment, the resulting system actions are more complex because they may involve the movement of file system ownership between systems in the shared HFS group. For more information about system actions that may occur in a shared HFS environment, see “Shared HFS implications during a planned shutdown of z/OS UNIX” on page 180.

Dynamically changing the BPXPRMxx parameter values

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

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

Example:

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

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

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

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

You can use the SETOMVS RESET command to dynamically add the FILESYSTYPE, NETWORK, and SUBFILESYSTYPE statements without having to re-IPL. However, if you change the values, a re-IPL will be necessary. For more information, see “Dynamically adding FILESYSTYPE statements in BPXPRMxx” on page 261.

Dynamically changing certain BPXPRMxx parameter values

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

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

Example: The following example shows you how to perform the calculations using the IPCMSGNIDS parameter, which determines the highest number of unique message queues in the system. To use SETOMVS IPCMSGNIDS=xxx to increase the current setting, you must calculate the highest number that you can specify. According to the description of IPCMSGNIDS in “IPCMSGNIDS and IPCSEMNIDS” on page 260, the formula is:

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

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

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

2. Compare 20000 with 4096 to find the smaller number (the MIN). 4096 is the smaller number.

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

```
SETOMVS IPCMSGNIDS=4096
```

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

MAXPROCSYS

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

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

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

MAXPTYs

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

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

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

IPCMSGNIDS and IPCSEMNIDS

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

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

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

IPCSHMNIDS and IPCSHMSPAGES

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

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

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

Dynamically switching to different BPXPRMxx members

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

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

Example:

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

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

Dynamically adding FILESYSTYPE statements in BPXPRMxx

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

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

```
DISPLAY OMVS,PFS
```

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

1. “Steps for activating the HFS file system for the first time”
2. “Steps for activating a single sockets file system for the first time” on page 262
3. “Steps for activating a multiple socket file system for the first time with Common INET” on page 263
4. “Steps for increasing the MAXSOCKETS value” on page 263
5. “Steps for adding another sockets file system to an existing Common INET (CINET) configuration” on page 264

Steps for activating the HFS file system for the first time

Perform the following steps to activate the HFS file system for the first time.

1. Set up a root HFS data set by putting the following statement in BPXPRMxx.

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

-
2. Create a temporary BPXPRMtt member that has the following statement.

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

-
3. Dynamically add the statements to BPXPRMtt.

```
SETOMVS RESET=(tt)
```

-
4. From TSO or the ISHELL, do the following:

- a. Unmount the current root file system.
 - b. Mount the root HFS data set as the new root file system.
 - c. Mount any additional HFS data sets as needed.
-
5. Add the following statements to the BPXPRMxx parmlib member that is used on IPL:
 - a. The FILESYSTYPE statement used in Step 2 on page 261.
 - b. A ROOT statement for the root HFS.
 - c. MOUNT statements for the additional mounts that should be done initially.
-

When you are done, you have activated the HFS file system for the first time.

Activating a single sockets file system for the first time

This section explains how to activate a single sockets file system for the first time. It uses the TCP/IP Socket File System for network sockets and also brings up support for local sockets.

Steps for activating a single sockets file system for the first time

Before you begin: You need to know what MAXSOCKETS value to use. The value used in the example may be different from the value that you want to use.

Perform the following steps to activate a single sockets file system for the first time.

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

```
/* Start Address Family AF_INET for Network Sockets */
FILESYSTYPE TYPE(INET) ENTRYPOINT(EZBPFINI)
NETWORK TYPE(INET) MAXSOCKETS(64000)
  DOMAINNAME(AF_INET) DOMAINNUMBER(2)

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

2. Dynamically add the statements to BPXPRM#.

```
SETOMVS RESET=(tt)
```

3. Restart TCP/IP.

```
S TCPIP
```

4. Add the parmlib statements in Step 1 to the BPXPRMxx member that is used on IPL.
-

When you are done, you have activated a single sockets file system for the first time.

Activating a multiple sockets file system for the first time with Common INET (CINET)

This section explains how to activate multiple sockets file systems for the first time with Common INET. In the example, two file systems are started: TCP/IP and AnyNet®.

Steps for activating a multiple socket file system for the first time with Common INET

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

Before you begin: You need to know what MAXSOCKETS value to use. The value used in the example may be different from the value that you want to use.

Perform the following steps to activate a multiple file system for the first time with Common INET (CINET).

1. Create a temporary BPXPRMtt member with the following statements:

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

-
2. Dynamically add the statements to BPXPRMtt.

```
SETOMVS RESET=(tt)
```

-
3. Restart TCP/IP.

```
S TCPIP
```

-
4. Start the Sockets Over SNA address space. (For more information, see *z/OS Communications Server: AnyNet Sockets over SNA*.)

-
5. Add the parmlib statements in Step 1 to the BXPRMxx member that is used on IPL.
-

When you are done, you have activated a multiple sockets file system for the first time with Common INET.

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

Increasing the MAXSOCKETS value

Steps for increasing the MAXSOCKETS value

The following steps assume that you want to make a permanent change to the MAXSOCKETS value without having to stop and then restart z/OS UNIX.

Restriction: This procedure can be used only if you have specified DOMAINNAME(AF_INET) or DOMAINNAME(AF_INET6). For AF_UNIX, you must update the MAXSOCKETS value in BPXPRMxx, then shut down and restart z/OS UNIX.

Before you begin: You need to know what MAXSOCKETS value you want to use, and you must have specified either DOMAINNAME(AF_INET) or DOMAINNAME(AF_INET6).

Perform the following steps to increase the MAXSOCKETS value.

1. Create a temporary parmlib member, BPXPRMtt, and add the following information:

```
NETWORK TYPE(INET) MAXSOCKETS(10000)
      DOMAINNAME(AF_INET) DOMAINNUMBER(2)
```

-
2. Activate the updated BPXPRMtt member.

```
SETOMVS RESET=(tt)
```

-
3. To make the change permanent, update the MAXSOCKETS value in the BPXPRMxx member that is used on IPL.
-

When you are done, you have increased the MAXSOCKETS value.

Tips:

1. You can add support for AF_INET6 to a running system for the first time. To do so, the NETWORK statement would specify DOMMAINNAME(AF_INET6) and DOMAINNUMBER(19). TCPIP would have to be recycled for this to take effect. You can add AF_INET6 in this way to an INET or as discussed in the next step to a CINET configuration.
2. You can change the MAXSOCKETS value for a CINET configuration with a similar procedure:
 - a. The TYPE() keyword of the NETWORK statement would specify the TYPE name of the CINET INET PFS, which was "CINET" in the previous examples.
 - b. INADDRANYPORT and INADDRANYCOUNT cannot be changed.

Adding another sockets file system to an existing Common INET (CINET) configuration

This section explains how to start a second TCP/IP sockets file system.

Steps for adding another sockets file system to an existing Common INET (CINET) configuration

Perform the following steps to add another sockets file system to an existing Common INET (CINET) configuration.

1. Create a temporary BPXPRMtt member with the following statements:

```
SUBFILESYSTYPE TYPE(CINET) NAME(TCPIP2) ENTRYPPOINT(EZBPFINI)
```

2. Dynamically add the statements to BPXPRMtt.

```
SETOMVS RESET=(tt)
```

3. Start the TCPIP2 address space.
-

4. Update the SUBFILESYSTYPE value in the BPXPRMxx member that is used on IPL.
-

When you are done, you have added another sockets file systems to an existing Common INET configuration.

Tracing events

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

Tracing events in z/OS UNIX

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

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

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

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

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

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

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

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

Tracing DFSMS/MVS events

You can also trace DFSMS/MVS events for the HFS.

Example: To set up a trace, issue::

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

or:

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

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

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

Re-creating problems for IBM service

Recommendation: If you are re-creating a problem for IBM service, you should increase the OMVS CTRACE buffer size to 8 MB.

Example: To increase the OMVS CTRACE buffer size to 8 MB:

```
TRACE CT,8M,COMP=SYSOMVS,PARM=CTnBPXxx
```

with the parmlib member specifying the desired options.

As an alternative, you can change the parmlib member to specify the desired buffer size. After you capture the dump for the problem, you can reset the trace buffer size to the original setting.

Example: To reset the trace buffer size to the original setting:

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

where xxxK is the size of the desired trace buffer.

Filtering trace data

Use the `JOBNAME=` parameter on the `TRACE CT` command to filter trace data. The `JOBNAME=` parameter can be used for the OMVS CTRACE to trace data just for jobs that run with the specific user ID or user IDs that are specified in the `JOBNAME` list. It is important to note that this filtering is based on the user ID of the job, not its jobname.

Displaying the status of the kernel or process

Display information about the kernel or processes as follows:

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

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

```
DISPLAY OMVS
```

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

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

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

For another example, see Figure 49 on page 251.

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

```
BPX0004I 10.17.23 DISPLAY OMVS 869
OMVS      ACTIVE 000E          OMVS=(XX,YY,ZZ)
```

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

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

```
D OMVS,PID=117440514

BPX0040I 14.16.58 DISPLAY OMVS 177
OMVS      000E ACTIVE          OMVS=(93)
USER      JOBNAME ASID        PID        PPID STATE  START    CT_SECS
MEGA      TC1      0021 117440514 117440515 HKI 14.16.14 .170
LATCHWAITPID= 0 CMD=ACEECACH
THREAD_ID  TCB#  PRI_JOB USERNAME ACC_TIME SC STATE
049614600000000 009E0438 .050 PTJ KU
04961D080000000 009D5E88 .002 SLP JSN
049625B00000000 009D8798 .003 SLP JSN
04962E580000000 009D5090 .012 SLP JSN
049637000000000 009D5228 .011 SLP JSN
04963FA80000000 009D5A88 .010 SLP JSN
049648500000000 009D8048 .011 SLP JSN
049650F80000000 009D81E0 .011 SLP JSN
049659A00000000 009D8378 .011 SLP JSN
049662480000000 009D8510 .011 SLP JSN
04966AF00000000 009D8930 .030 SLP JSN
```

Selected threads can be canceled.

Example:

```
F BPX0INIT,FORCE=117440514.04962E5800000003
BPXM027I COMMAND ACCEPTED.

F BPX0INIT,TERM=117440514.0496624800000009
BPXM027I COMMAND ACCEPTED.
```

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

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

Displaying the status of BPXPRMxx parmlib limits

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

```
DISPLAY OMVS,L
BPX00511 14.05.52 DISPLAY OMVS 904
OMVS 0042 ACTIVE OMVS=(69)
SYSTEM WIDE LIMITS: LIMMSG=NONE
```

	CURRENT USAGE	HIGHWATER USAGE	SYSTEM LIMIT
MAXPROCSYS	1	4	256
MAXUIDS	0	0	200
MAXPTYS	0	0	256
MAXMAPAREA	0	0	256
MAXSHAREPAGES	0	10	4096
IPCMGNIIDS	0	0	500
IPCSEMNIDS	0	0	500
IPCSHMNIDS	0	0	500
IPCSHMPAGES	0	0	262144 *
IPCMGQBYTES	---	0	262144
IPCMGQNUM	---	0	10000
IPCSHMPAGES	---	0	256
SHRLIBRGNSIZE	0	0	67108864
SHRLIBMAXPAGES	0	0	4096

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

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

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

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

Taking a dump of the kernel and user processes

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

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

The steps are:

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

3. Take the dump.
4. Review the dump completion information.

Displaying the kernel address space

Example: To find the kernel address space and associated data spaces, issue:

D A, OMVS

Result: The output will be similar to the following:

```
D A,OMVS
IEE115I 12.55.47 94.208 ACTIVITY 503
JOBS      M/S      TS USERS      SYSAS      INITS      ACTIVE/MAX VTAM
00001     00013     00002     00019     00019     00002/00050
OMVS      OMVS      OMVS      NSW SO    A=000E    PER=NO     SMC=000
          CT=033.466S  ET=03.44.48
          WUID=STC06055 USERID=OMVSKE
          ADDR SPACE ASTE=0173ECC0
          DSPNAME=SYSZBPXU ASTE=00A35
          DSPNAME=SYSGFU01 ASTE=007F8
          DSPNAME=SYSZBPX3 ASTE=007F8
          DSPNAME=SYSIGWB1 ASTE=007F8
          DSPNAME=SYSZBPX2 ASTE=00A35
          DSPNAME=SYSZBPX1 ASTE=00A35
```

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

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

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

Displaying process information

Example: To display the process information for address spaces, issue:

D OMVS,A=ALL

Result: The output will be similar to the following:

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

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

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

Displaying global resource information

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

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

Preallocating a sufficiently large dump data set

Because you are dumping multiple address spaces, multiple data spaces, and multiple storage data areas, you may need a much larger dump data set defined than is normally used for dumping a single address space. You should preallocate a very large `SYS1.DUMPnn` data set. For more information on `SYS1.DUMPnn` data, see the `DUMPDS` command in *z/OS MVS System Commands*.

`SDUMP` has a limit on how much storage it allows in a single dump. It is called `MAXSPACE`. To determine the current value of `MAXSPACE`, issue the `D D,0` command. The default value is 500 megabytes. To change this value, issue:

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

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

Taking dumps

To initiate the dump, enter this command:

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

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

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

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


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

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

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

This example specifies data spaces:

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

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

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

Reviewing dump completion information

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

Recovering from a failure

The operator needs to recover if a failure occurs:

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

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

z/OS UNIX system failure

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

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

File system type failure

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

If a file system type fails to initialize, the system normally issues message BPXF006I. If the failing file system type was specified with the option to prompt for restart (the default), the error that caused the problem can be corrected, and then the prompt responded to. If it was specified with the option to not prompt for restart,

the system continues to run without that file system type, but requests to use the files in any file systems of that file system type will fail.

In rare cases, the initialization of a file system type may fail due to a severe programming or environmental error, such as a severe storage shortage in the kernel address space. The failure can occur before the file system type is initialized, and, on rare occasions, the BPXF006I message will not be issued. In these cases, the severe programming or environmental error should be addressed first. After the severe condition that prevented the initialization of the file system type is resolved, the operator can manually initialize the file system type with the SETOMVS RESET=xx system command. For more information about the SETOMVS RESET=xx command, see *z/OS MVS System Commands*.

File system failure

These events can be symptoms of file system failure:

- 0F4 abend
- EMVSPFSFILE return code
- EMVSPFSPERM return code
- A file becomes unrecognizable or unopenable

After a failure of a file system, the operator:

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

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

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

Recovery of DCE components

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

Managing Interprocess Communication (IPC)

Users can invoke applications that create IPC resources and wait for IPC resources. IPC resources are not automatically released when a process is ended 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

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

```
ipcs -w
```

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

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

```
ipcs -w
```

Chapter 15. Managing processing for z/OS UNIX

Overview

This chapter discusses the tasks involved with the managing of a z/OS system with z/OS UNIX services.

Controlling printing

Control printing by doing the following:

- Designate printers to be used for shell users and applications
- Set up default printers for each user
- Control output print separators

If desired, you can arrange for all printing to be done by one or two printers by assigning one or more output classes for all users. Then you and the users can look at the printer queues for those output classes to check for all output.

If the OS/390 Print Server feature has been enabled, it will provide an alternate version of the **lp** command, as well as related utilities. For more information, see *OS/390 Print Server User's Guide for OS/390 UNIX System Services*, SC54-5543-01.

Designating printers

Tell the application programmers the destinations or symbolic names for printers you specified in JES initialization statements. The *dest* option of the **lp** command uses the same destinations as the DEST parameter in the OUTPUT JCL statement.

The *dest* option on **lp** can be:

- LOCAL for any installation printer.
- A destination that is defined in a JES2 DESTID initialization statement.
- Omitted. The system uses the default printer.

For information about DESTID, see *z/OS JES2 Initialization and Tuning Reference*.

Setting up default printers

Each user has a number of default printers specified in different ways, as shown in Table 16. The system will use printer number 1, if designated; if not, the system will use printer number 2; and so on.

Table 16. Default printers

Printer Number	Printer Designation	Specified by
1	The printer in the <i>dest</i> option of the lp shell command, or the <code>printf()</code> or <code>fprintf()</code> functions.	User or application programmer
2	The printer LPDEST environment variable	User or system programmer
3	The printer PRINTER environment variable	User or system programmer
4	The printer in the RACF user profile. It is specified by the DEST parameter of the RACF ADDUSER or ALTUSER command.	Security administrator

Controlling output print separators

JES controls the print separators, also called *cover pages* and *banner pages*, for SYSOUT output for all users, including z/OS UNIX users.

To place a user's name and address in the print separator for forked processes, specify the user's name and address in the WORKATTR segment of the RACF user profile. See "Defining z/OS UNIX users to RACF" on page 60.

Controlling code page conversion

For an overview of character sets and code pages, refer to *National Language Support Reference Manual, Volume 2*, SE09-8002.

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

The z/OS UNIX Application Services can process data in the following code pages:

- Any of the **EBCDIC Latin 1 Country-Extended Code Pages**
- **Japanese (Latin) Extended Code Page 01027**, which defines singlebyte encodings for character set 01172 (Japanese Extended EBCDIC/PC Common)
- **Japanese Combined Code Page 00939**, which is the combination of code page 01027 and code page 00300. Code page 00300 (Japan [Kanji]–Host, DBCS) defines DBCS encodings for character set 00370 (IBM Japanese Graphic Character Set, Kanji)

Data intended for processing by the z/OS shell may require conversion to one of the preceding code pages. This data may be encoded in:

- **Latin 1 code page 00500**, which is used for Systems Application Architecture® (SAA®).
- An **ASCII code page**, for example, for a file from a workstation. A source program on a tape archive (TAR) tape might be stored in the ASCII code set.
- **Code page 00293**, which the C/C++ Compiler can optionally use.
- **Code page 00290**, Japanese (Katakana) singlebyte.
- **Code page 00930**, the Japanese combined code page (code page 0290 plus DBCS code page 300).

For code page 00037, only two characters are different from code page 01047:

- Right square bracket (])
- Left square bracket ([)

If you have characters from the preceding list in your data, you need to convert from one code page to another when, for example:

- Transferring files between a workstation and the file system.
- Copying data between an MVS data set and an HFS file.
- Placing data in SYSOUT data sets.
- Passing JCL pathname data to programs, unless the name contains only characters in the portable filename character set.
- Passing JCL parameters and pathnames to a shell invoked from a batch program, unless the parameters and names contain only characters in the portable filename character set.

- Using the **lp** command to print. You need to convert the data before sending it to the printer.
- Using the **pax** command.

Considerations for data conversion

Converting singlebyte data

If MVS data is singlebyte, you can specify the conversion at the same time that you copy the data.

Users need to specify the CONVERT operand in the TSO/E OCOPY, OGET, OPUT, OGETX, and OPUTX commands to convert the data that the command is copying. Copying can be from data sets or HFS files and to data sets or HFS files.

Converting doublebyte data

Doublebyte data that is already in a supported DBCS code page, such as IBM-939, does not need to be converted. DBCS data not in code page IBM-939 must be converted to IBM-939 with the **iconv** command, so that it can be processed in the z/OS UNIX environment.

If the data is in a code page not supported by the shell, you can copy the data with the OCOPY command first and then convert it using the **iconv** command. Or you can convert the data with the TSO/E ICONV CLIST and then copy it using the OCOPY command.

For more information on code page conversion, see:

- *z/OS C/C++ User's Guide*
- *z/OS UNIX System Services User's Guide*

Using character conversion tables

IBM supplies the following character conversion tables as members in SYS1.LINKLIB:

BPXFX100

Null conversion

BPXFX111

Code page 00037 to and from 01047 for non-APL 3270

BPXFX211

Code page 00037 to and from 01047 for APL 3270

BPXFX311

ASCII code page 008859 to and from 01047

In particular, for the OMVS command, BPXFX100 is the default conversion table. As shipped, FSUMQ000 is an alias for BPXFX100. To change the OMVS default table, move the FSUMQ000 ALIAS to the new default, or rename the new default table to FSUMQ000.

Users who need different conversion tables can manually override the default table by using the CONVERT operand. In addition, the system programmer can write a REXX exec or CLIST to invoke the OMVS command with the proper table.

The source for these members is shipped in SYS1.SAMPLIB. If you need to change them, see "Customizing code page conversion" on page 278.

Example of data conversion specified by a user: The following example uses the OPUT command for conversion of singlebyte data. This command allows you to

copy a sequential data set or partitioned data set member to an HFS file. Code page conversion is an option with this command.

```
OPUT WORKLOAD.TOTALS(OCT17) 'u/turbo/wkld/totals/oct17' TEXT CONVERT(YES)
```

In this example, the user ID TURBO copied a member from a PDSE into a file. The partitioned data set member OCT17 was copied from the data set TURBO.WORKLOAD.TOTALS to a text file with the pathname **/u/turbo/wkld/totals/oct17**.

Data was converted from the z/OS UNIX country-extended code page to code page 01047, using the default conversion table because YES was specified. To use a different conversion table, specify its name—for example, BPXFX311 for conversion from the ASCII conversion table. If you do not want conversion, omit the CONVERT operand.

Customizing code page conversion

If the installation has special conversion needs for singlebyte data conversion, create the needed character conversion table by customizing an existing table:

1. Copy the assembler source for a table from SYS1.SAMPLIB into a new data set.
2. Edit the new data set to customize the table. SYS1.SAMPLIB(BPXFX100) shows the format of the conversion tables and, in its prolog, gives instructions and examples of how to edit a table.
3. Assemble the table.
4. Link-edit the table into a load module in SYS1.LINKLIB or another partitioned data set.

Each table is 1792 bytes long and contains the 8-bit codes that the system substitutes for characters in the input data set or file. Each table contains nine sections; you may have to change the data in all nine sections. Each member has a TO and a FROM subtable:

- The TO subtable is used to translate data from another code page to 01047.
- The FROM subtable is used to translate data from code page 01047 to another code page.

Example of code page conversion of OMVS command

For customization, suppose you change a table, name it WCOFXCHG, and place it in a SYS1.LINKLIB member. Then you would use the following OMVS command to invoke the shell, with SYS1.LINKLIB understood as the location of WCOFXCHG:

```
OMVS CONVERT((WCOFXCHG))
```

To avoid conflicts in the names of modules containing tables, begin your name with letter *K* through *Z*; letters *A* through *J* are reserved for IBM use.

Managing z/OS UNIX in relation to other processing

JES processing and certain applications are affected by z/OS UNIX.

JES2 processing

In a JES2 multi-access spool (MAS) complex, a z/OS UNIX application may experience one or both of these conditions:

- **The system can convert the job on one system and interpret it on another.** If all systems do not have z/OS UNIX, the job processing can begin on a system with z/OS UNIX installed and started, but continue on a system without z/OS UNIX installed.

If a job requires z/OS UNIX or a file system, use a JES2 /*JOBPARM statement with a SYSAFF keyword to direct the job to the correct system.

If you need to bring down JES2, there may still be a number of initiators that are provided by WLM for use on fork and spawn. These initiators time out after 30 minutes on their own. To terminate the initiators, you can issue the following operator command:

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

For more information, see “Partial shutdowns (for JES2 maintenance)” on page 254.

JES3 processing

In a JES3 global/local configuration, a z/OS UNIX application may experience one or both of these conditions:

- **One system can complete conversion and interpretation and another system can run the job.** If all systems do not have z/OS UNIX, the job might be assigned to a system without z/OS UNIX. In this case, the job will fail. To prevent this problem, use a JES3 /*MAIN statement with a SYSTEM keyword to direct the job to a system with z/OS UNIX.

Applications processing

The following applications cannot request z/OS UNIX services:

- **Customer Information Control System/ESA (CICS/ESA®):** A CICS transaction cannot access a file in a file system, because the access would put CICS in a wait. Also, other z/OS UNIX functions could not be used and would abnormally end CICS.
- **Information Management System/ESA (IMS/ESA®):** An IMS application cannot access a file in a file system, because the access would put the IMS application in a wait. Also, other z/OS UNIX functions could not be used and would abnormally end the IMS application.

The IMS batch message processing program (BMP) can request z/OS UNIX services.

Accessing the Language Environment run-time library

For most z/OS UNIX applications, the Language Environment run-time library (SCEERUN and SCEERUN2) is needed. The SCEERUN and SCEERUN2 data sets can be placed in LNKST, accessed via STEPLIB, or managed by Run-Time Library Services (RTLS).

When choosing a method for run-time library access, you should consider the following:

- Can the Language Environment run-time library be placed in LNKST, without adversely affecting other applications?
- Is the Language Environment run-time library heavily used at your installation?
- Does the RTL require frequent testing or replacement with new versions?

Placing the Language Environment run-time library (SCEERUN and SCEERUN2) in LNKLST requires the least amount of setup. If this method is used, then consider placing the SCEELPA data set (which contains key modules) in LPA for better performance. See “Improving performance of run-time routines” on page 333.

However, 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:

1. 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.
2. Add the SCEERUN data set to your TSO/E logon procedure by concatenating it to the ISPLLIB DD statement (if it exists) and then concatenating it to the STEPLIB DD statement (if it exists). You can also use the TSOLIB function to add the SCEERUN data set. After you have added the SCEERUN data set, the TSO/E OMVS command can begin to use it.
3. Add the following statement to the **/etc/rc** file:

```
export STEPLIB=h1q.SCEERUN
```

This will be used by daemons started in **/etc/rc**.

4. In **/etc/profile**, remove:

```
if [ -z "$STEPLIB" ] && tty -s;
then
    export STEPLIB=none
    exec sh -L
fi
```

and replace with

```
export STEPLIB=h1q.SCEERUN
```

This is used when issuing commands and utilities in the shell environment. If a small number of interactive users need a special version of the run-time library, the STEPLIB environment variable can be set in the **\$HOME/.profile** for each of these users.

5. Add the SCEERUN data set on a STEPLIB DD statement to any job invoking BPXBATCH.
6. Add the SCEERUN data set to the STEPLIBLIST statement of the BPXPRMxx parmlib member.
7. The SCEERUN data set will also need to be APF-authorized.
8. It is recommended that the SCEERUN2 data set be placed in LNKLST, even though the SCEERUN data set is accessed through STEPLIB. Because the SCEERUN data set does not contain module names that conflict with pre-Language Environment run-time libraries, adding it to LNKLST will not have any adverse effects.

The STEPLIB approach does have a performance overhead which can be avoided if RTLS is used. RTLS allows you to access the Language Environment run-time library without using STEPLIB or without placing Language Environment in LNKLST. RTLS also enables you to manage multiple levels of the Language Environment run-time libraries and allows access to specific levels, with the use of Language

Environment run-time options. For information on how to set up RTLS, see *z/OS Language Environment Customization* and *z/OS MVS Initialization and Tuning Reference*.

In addition, to the parmlib setup of RTLS, you must also specify the RTLS(ON), LIBRARY and optionally VERSION run-time options in the RUNOPTS parameter of BPXPRMxx.

Example:

```
RUNOPTS(RTLS(ON) LIBRARY(xxxxxxxx) VERSION(yyyyyyyy))
```

where xxxxxxxx is the library name and yyyyyyyy is the version name assigned in the RTLS parmlib member for the specific level of Language Environment run-time needed.

For BPXBATCH processing, you still have to specify the SCEERUN data set on a STEPLIB DD statement even though the RUNOPTS parameter has been set in the BPXPRMxx member.

Enabling the fastpath support for system authorization facility (SAF)

When the BPX.SAFFASTPATH FACILITY class profile is defined, the security product is not called if z/OS UNIX can quickly determine that file access will be successful. When the security product is bypassed, better performance is achieved, but successful file accesses cannot be audited. If the security product is called, it is still possible that access will be successful, and that audit records will be created; for example, when the permission bits do not grant access, but UNIXPRIV authority, or an access control list, does. Be aware that auditing successful accesses can generate enormous amounts of audit records, particularly for directory searches.

- If the BPX.SAFFASTPATH FACILITY class profile is defined when the system is IPLed, the SAF fastpath support is enabled.
- If it is defined after the system is IPLed, you must issue the SETOMVS or SET OMVS operator command to activate the fastpath support.

You can also start the refresh by issuing the following command, where xx represents a BPXPRMxx member that is empty:

```
SET OMVS=xx
```

Users do not need to be permitted to the BPX.SAFFASTPATH profile.

To define the BPX.SAFFASTPATH profile, issue the following RACF command:

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

Tip: If your installation uses the IRRSXT00 exit to control HFS access, do not define the BPX.SAFFASTPATH profile.

Determining problem causes

If a problem occurs, the system may write a dump and issue messages or an abend. Collect the problem data and determine the cause of the problem as you would for any system problem. For information about how to take a dump, see “Taking a dump of the kernel and user processes” on page 268.

Abends

z/OS UNIX services issues system completion codes: abend codes EC6 and 422.

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.

Some abends are a normal result of a **kill** shell command, an **exec** shell command or program function, or the ending of a process. Others are caused by errors.

Return codes and reason codes

If a z/OS system service fails, a failing return code and reason code is set. Reason codes are unique and should supply enough information to debug the problem. You can set a slip trap on a specific reason code to gather further diagnostic data. For information about setting the slip trap, see the section on reason codes listed by value in *z/OS UNIX System Services Messages and Codes*.

Messages

z/OS UNIX issues messages with the following prefixes:

BPX Messages from the System Services component

FDBX Messages from the dbx debugger

FOM Messages from the Application Services component

FSUM Messages from the Shell and Utilities

DFSMS/MVS issues messages with the prefix IGD.

Messages to the operator and system programmer have identifiers; messages from the shell to the interactive user do not have identifiers.

You can use the `_BPXK_JOBLOG` environment variable to specify that messages be written to a job log in the HFS.

A quick and easy way to look up message explanations is by using LookAt. For more information, see “Using LookAt to look up message explanations” on page xx.

Component identifiers

The component identifiers that are used in dumps and symptom strings are:

Component Identifier	Code	Module Prefix
DF185	HFS file system adapter	GFU
SCPX1 SCPX4 SCPX6	z/OS UNIX System Services	BPX FOM BOP
SCPX2	Shell and Utilities, shell initialization, TSO/E OMVS command, and the c89 shell command	FSUM
SCPX3	dbx debugger	FDBX

Dump formatting

To format problem data in a stand-alone dump or SVC dump, use the interactive problem control system (IPCS) OMVSDATA subcommand. OMVSDATA is not useful in an SYSMDUMP dump or a core dump, which the system writes for an application program, because these dumps do not contain the z/OS UNIX programs or data structures.

See the following:

- *z/OS MVS System Codes* for the abend codes
- *z/OS MVS IPCS Commands* for the syntax of the IPCS OMVSDATA command
- *z/OS MVS Diagnosis: Reference* for formatting an SVC dump
- *z/OS MVS Diagnosis: Reference* for general problem determination procedures
- *z/OS MVS Diagnosis: Tools and Service Aids* for SYSMDUMP and SYSABEND dumps produced by applications

Diagnosing problems in an IBM-supplied z/OS UNIX program

If the problem is in the shell or debugger, the system treats it as an application problem.

To prevent a CEEDUMP when you want to get a SYSMDUMP, you need to tell Language Environment to abend without doing any of its termination activities, such as CEEDUMP. To do this, the application must invoke the CEE3ABD callable service (which terminates the enclave with an abend) with the clean-up parameter set to zero. There are suboptions for the TERMTHDACT suboption that enables you to specify the amount of dump data that is to be collected. See *z/OS Language Environment Programming Guide*.

You can also specify a dump by specifying a value for the `_BPXK_MDUMP` environment variable. If that environment variable is set, you do not have to allocate a SYSMDUMP data set for the TSO/E session. The dump will be written to either the MVS data set or the specified HFS file.

If the `_BPXK_MDUMP` environment variable is not set, then you can specify a dump by allocating a SYSMDUMP data set for the TSO/E session. The system:

- Creates a file in the user's working directory.
- Names it **coredump.pid**, where **pid** is the process ID for the process being dumped. It is in hexadecimal format.
- Writes a core dump in the file. The core dump is a SYSMDUMP dump.

To use the core dump, do the following:

1. To copy the file into an MVS data set with a record length (LRECL) of 4160, enter a TSO/E OGET command.
2. To analyze the dump, use IPCS. It is recommended that you use the IPCS subcommands STATUS and SUMMARY FORMAT CURRENT.

You can dynamically request a SYSMDUMP by using the SIGDUMP signal; for more information, see *z/OS C/C++ Programming Guide*. Use the `_BPXK_MDUMP` environment variable to specify where the SYSMDUMP is to be written to.

You can also request a SYSMDUMP by using the `F BPXOINIT,DUMP=` operator command. For both the SIGDUMP signal and the `F BPXOINIT,DUMP` command, the `_BPXK_MDUMP` environment variable must be set to an MVS data set name. If it is set to an HFS filename or defaulted to OFF, then both the SIGDUMP signal and the `F BPXOINIT,DUMP` command may be ignored.

Diagnosing problems in application programs

The dbx debugger helps in debugging application programs written in the C language. With the debugger, the application programmer can set breakpoints at source statements and function entry points, display and modify storage using program variable names rather than absolute storage addresses, trace execution at the source statement level, and so on.

Diagnosing hangs during z/OS UNIX initialization

If there is a hang during z/OS UNIX initialization, the hang is likely to occur during the initialization process (**/etc/init** and **etc/rc**).

If you receive message BPXP006E indicating that z/OS UNIX is being initialized, you can check the **/etc/log** file to see what the last command processed from **/etc/rc** was. This may help you determine the cause of delay or hang.

(The sample **/etc/rc** file that is shipped with z/OS UNIX includes the `set -v -x` command. That command specifies that shell input lines are to be printed to **/etc/log** as they are run, in addition to commands and their arguments.)

If you are a superuser (permission to BPX.SUPERUSER is not sufficient), you can view **/etc/log** during a hang in **/etc/rc** by starting a shell from a superuser and issuing the following command:

```
cat /etc/log
```

Chapter 16. Managing a temporary file system (TFS)

Overview

This chapter discusses the temporary file system (TFS). It is an in-memory physical file system that supports in-storage mountable file systems. Normally, a TFS runs in the kernel address space, but it can be run in a logical file system (LFS) colony address space. For more information, see “Running a physical file system in a colony address space” on page 43.

The TFS supports attributes unique to z/OS UNIX (for example, external symbolic links and the create time attribute) and is not part of a branded product.

Recommendations:

1. If more space is needed than can fit in an address space, you should always run TFS in a colony. When TFS is run in a colony, you can use the STOP and MODIFY commands. The MODIFY command has the parameters STOP, TERM, FORCSTOP, and FORCETERM.
2. Because TFS uses virtual storage, make sure that your real and auxiliary storage configurations are enough to accommodate the size of all of the TFS file systems that you mount.

In this chapter

This chapter covers the following subtasks.

Subtasks	Associated procedure (see . . .)
Mounting the TFS	“Steps for mounting the TFS” on page 286

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

- “Comparing UNIX security and z/OS UNIX security” on page 290.
- “Establishing the correct level of security for daemons” on page 291.

See *z/OS C/C++ Programming Guide* for more information about writing a daemon program.

Maximum file size

The maximum file size that TFS will support is a function of the TFS block size. The following chart shows the approximate maximum file sizes based on the block size factor:

0	2 GB
1	25 GB
2	240 GB
3	2 TB
4	17 TB

Mounting the TFS

The TFS is automatically mounted if the kernel is started in minimum mode. In this environment, the TFS is the in-storage file system and it defaults to the root file system. Because it is an in-storage file system, the TFS delivers high-speed I/O.

If you are using kernel services in full function mode with a hierarchical file system, you may want to mount a TFS over **/tmp**. If you do, it can be used as a high-speed file system for temporary files. However, you cannot recover **vi** files if the system goes down because **vi** writes temporary files to TMPDIR (**/tmp** by default). To recover these files, you can use the **exrecover** command, which automatically runs from **/etc/rc**.

Recommendation: The administrator can use the TMP_VI environment variable to specify that the temporary files be written to a specified directory instead of TMPDIR.

Steps for mounting the TFS

If the kernel is started in minimum setup mode, the TFS is automatically mounted. If the TFS is to be used in other situations, it is made available by mounting. You can mount it with a MOUNT or ROOT parmlib statement or via one of the supported mount commands.

Base your choice on the method that you want to use.

If you are using . . .	Then . . .
The FILESYSTYPE parmlib statement	You must specify the TYPE as TFS. Example: <pre>FILESYSTYPE TYPE(TFS) ENTRYPPOINT(BPXTFS)</pre>
The MOUNT statement or command	• TYPE must be TFS • FILESYSTEM must be a unique name for the file system. Recommendation: Specify the pathname that it is mounted on. This may make it easier to understand the output produced by commands such as df. • MODE can be either RDWR or READ. • PARM specifies the amount of virtual storage the file system uses. It also specifies other information as listed in Table 17 on page 287. PARM is specified as PARM(' -s n ') where <i>n</i> is the approximate size in megabytes. If PARM is omitted or is not valid, the TFS defaults to 1MB. If the mount request specifies a size in megabytes that is too large for the address space, the request will fail with an EMVSERR (9D). Try the request again, using a smaller value. Example: A sample MOUNT statement for a 10MB in-storage file system mounted over /tmp: <pre>MOUNT FILESYSTEM('/TMP') TYPE(TFS) MOUNTPPOINT('/tmp') PARM(' -s 10')</pre>

You can now perform the steps for the decision you have made.

Restriction: You cannot mount a TFS by using a DDname.

Options for the PARM parameter

The PARM parameter has the following options:

Table 17. Options for the PARM parameter

Option	Description										
-s <size>	<size> is the number of MB for the file system. If the value specified is larger than the size that can be supported, the maximum size will be used. Default: 1										
-b <block>	<block> specifies the blocking factor used to set the size of a TFS block. The following chart shows how <block> relates to the TFS block size: <table><tr><td>0</td><td>4K</td></tr><tr><td>1</td><td>8K</td></tr><tr><td>2</td><td>16K</td></tr><tr><td>3</td><td>32K</td></tr><tr><td>4</td><td>64K</td></tr></table> Default: 0	0	4K	1	8K	2	16K	3	32K	4	64K
0	4K										
1	8K										
2	16K										
3	32K										
4	64K										
-c <cache>	<cache> is the amount of buffer storage in MB that TFS will use in the 32-bit address range to support a 64-bit TFS. This parameter is ignored for file systems that are allocated in the 31-bit range. The default is calculated based on the TFS block size such that the numbers of cache buffers is 256.										
-u <uid>	<uid> is the numeric UID that is to be assigned to the root directory of the file system. Default: 0										
-g <group>	<group> is the numeric GID that is to be assigned to the root directory of the file system. Default: 0										
-p <perm>	<perm> is the permission bits in octal to be assigned to the root directory of the file system. Default: 0777										
-3	Specifies that TFS is to allocate the file system in 31-bit mode regardless of system capabilities.										

Invalid mount options are ignored.

Unmounting the TFS

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 dot (.) and dot-dot (..) and nothing else.

Using a TFS in a shared HFS environment

A TFS can be used in a shared HFS environment. If you are using a TFS for **/tmp**, the FILESYSTEM name must be different because each system will require its own copy.

Example: The MOUNT statement would need to be:

```
MOUNT FILESYSTEM('/TMP&SYSNAME. ')  
TYPE(TFS) MODE(RDWR) NOAUTOMOVE  
MOUNTPOINT('/&SYSNAME./tmp')  
PARM('-s 10')
```

Because &SYSNAME is different on each system, '/TMP&SYSNAME.' will have a different file system name on each system.

| Read-only BPAM access

| You can set up read-only basic partitioned access method (BPAM) access to TFS
| files. Each TFS directory is treated as if it were a PDSE or PDS directory. For more
| information about BPAM, see *z/OS DFSMS: Using Data Sets*.

Chapter 17. Setting up for daemons

Overview

This chapter discusses the steps for setting up for daemons and their security levels. A *daemon process* is a process that runs in the background and is not associated with any particular terminal or user. Daemons have superuser authority and can issue authorized functions such as `setuid()`, `seteuid()` and `spawn()` to change the identity of a user's process.

In this chapter

This chapter covers the following subtasks.

Subtasks	Associated procedure (see . . .)
Establishing the correct level of security for daemon	"Steps for preparing the security program for daemons" on page 294
Customizing the system for IBM-supplied daemons	"Steps for defining programs from load libraries to program control" on page 296 "Step for defining programs in HFS files to program control" on page 297 "Steps for setting up HFS control" on page 297 "Steps for setting up enhanced program security" on page 300
Customizing the system for IP-supplied daemons	"Steps for customizing the system for IP-supplied daemons" on page 302
Customizing the IBM-supplied daemons	"Steps for customizing the inetd daemon" on page 303 "Steps for customizing the uucpd daemon" on page 303 "Step for customizing the rlogind daemon" on page 304
Setting up security procedures for daemons	"Steps for setting up security procedures for daemons" on page 308

Subtasks	Associated procedure (see . . .)
Tracking down problems when setting up daemons and servers	“Steps for obtaining security information about users” on page 67 “Steps for obtaining security information about a group” on page 67 “Steps for verifying that the sticky bit is on” on page 312 “Step for using external links to access MVS load libraries” on page 313 “Steps for finding modules that were not defined to program control” on page 313 “Step for checking the daemon authority” on page 315 “Step for checking the server authority” on page 315 “Step for checking the SURROGAT class profile” on page 316
Setting up for rlogin	“Steps for setting up for rlogin” on page 317

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

- “Comparing UNIX security and z/OS UNIX security.”
- “Establishing the correct level of security for daemons” on page 291.

See *z/OS C/C++ Programming Guide* for more information about writing a daemon program.

Comparing UNIX security and z/OS UNIX security

Some of the people who perform the tasks described in this document have a background in MVS, while others have experience in UNIX systems other than z/OS UNIX.

MVS, traditional UNIX, and z/OS UNIX systems manage user identities differently. Table 18 contrasts various aspects of security on these systems.

Table 18. Comparing traditional UNIX, MVS, and z/OS UNIX security

CATEGORY	Traditional UNIX	MVS	z/OS UNIX
User identity	Users are assigned a unique UID, a 4-byte integer and user name.	Users are assigned a unique user ID of 1-to-8 characters.	Users are assigned a unique user ID with an associated UID.
Security identity	UID	User ID	UID for accessing traditional UNIX resources and the user ID for accessing traditional z/OS resources
Login ID	Name used to locate an UID	Same as the user ID	Same as the user ID

Table 18. Comparing traditional UNIX, MVS, and z/OS UNIX security (continued)

CATEGORY	Traditional UNIX	MVS	z/OS UNIX
Special user	Multiple user IDs can be assigned an UID of 0.	RACF administrator assigns necessary authority to users.	Multiple user IDs can be assigned an UID of 0 or users can be permitted to BPX.SUPERUSER.
Data set access	Superuser can access all files.	All data sets controlled by RACF profiles.	Superuser can access all HFS files; data sets controlled by RACF profiles.
Identity change from superuser to regular user	Superuser can change the UID of a process to any UID using <code>setuid()</code> or <code>seteuid()</code> functions.	APF-authorized program can invoke SAF service to change identity.	There are two options: 1. If the BPX.DAEMON FACILITY class profile is not defined, the superuser can change the UID of a process to any UID using <code>setuid()</code> or <code>seteuid()</code> functions. 2. Or, the superuser must be permitted to the BPX.DAEMON facility class profile in order to change UIDs.
Identity change from regular user to superuser	su shell command allows change if user provides root's password.	No provision for unauthorized user to change identity.	su shell command allows change if the user is permitted to the BPX.SUPERUSER FACILITY class profile or if the user provides the password of a user with a UID of 0.
Identity change of a regular user from one UID to another UID	su shell command allows change if user provides password.	No provision for unauthorized user to change identity.	su shell command allows change if user provides password.
Terminate user processes	Superuser can kill any process.	MVS operator can cancel any address space.	Superuser can kill any process.
Multiple logins	Users can login to a single user ID multiple times.	Users can only log on to TSO/E once per user ID.	Users can rlogin multiple times to a single user ID and logon once to TSO/E at the same time.
Login daemons	inetd , rlogind , lm , and telnetd process user requests for login. A process is created with the user identity (UID).	TCAS and VTAM process user requests for logon. A TSO/E address space (process) is created with the user identity (user ID).	Users can log on to TSO/E or login using one of the login daemons. In all cases, an address space is created with both an MVS identity (user ID) and an UID.

Establishing the correct level of security for daemons

Kernel services support two levels of appropriate privileges: UNIX level and z/OS UNIX level. This lets you distinguish superusers from daemons. Read the following descriptions to help you decide which level of security is appropriate for your installation.

UNIX level

If the BPX.DAEMON FACILITY class is not defined, your system has UNIX-level security. In this case, the system is less secure.

This level of security is for installations where superuser authority has been granted to system programmers. These individuals already have permission to access critical data sets such as PARMLIB, PROCLIB, and LINKLIB. These system programmers have total authority over a system.

Programs that run with superuser authority have daemon level authority. They can issue MVS identity-changing services such as `setuid()`, `seteuid()` and `__spawn()` without having first issued a successful `_passwd()` for the target user ID.

To use the UNIX level of security, assign UID(0) to the superuser. Also assign UID(0) to the user ID used for running daemon programs; for example, **inetd** or **cron**.

z/OS UNIX level

There are two z/OS UNIX levels:

- RACF running with enhanced program security, BPX.DAEMON defined, and BPX.MAINCHECK defined. You can use BPX.MAINCHECK for any privileged z/OS UNIX application that requires a program controlled environment, because the application uses a privileged z/OS UNIX service that requires one. An example is the `__passwd()` service, which is used by applications such as telnet and rlogin.
- BPX.DAEMON FACILITY

RACF with enhanced program security, BPX.DAEMON, and BPX.MAINCHECK

If you enable enhanced program security, and you have any daemons or servers that run execute-controlled programs (MVS programs defined to RACF in the PROGRAM class using EXECUTE authority, or loaded from libraries using EXECUTE authority), then you must define the initial program executed by your daemon or server as a trusted ("MAIN") program to RACF via the PROGRAM class. If this initial program resides in the z/OS UNIX file system, rather than in an MVS library, you will need to move it to an MVS library.

Additionally, you can choose whether to extend the enhanced program security protection to your UNIX daemons and servers that do not make use of RACF execute-controlled programs. You would enable this function by defining the profile BPX.MAINCHECK to RACF in the FACILITY class. Again, you would need to ensure that the initial program executed by your daemon or server resides in an MVS library and you would need to define it to RACF as a PROGRAM with the MAIN attribute.

Kernel services that change a caller's z/OS user identity require the target z/OS user identity to have an OMVS segment defined. If you want to maintain this extra level of control at your installation, you will have to choose which daemons to permit to BPX.DAEMON. . You will also have to choose the users to whom you give the OMVS security profile segments. To accomplish this, refer to "Steps for preparing the security program for daemons" on page 294.

"Steps for setting up enhanced program security" on page 300 explains how to set up enhanced program security.

BPX.DAEMON

If the BPX.DAEMON profile in the FACILITY class is defined, your system has z/OS UNIX security. Your system can exercise more control over your superusers.

This level of security is for customers with stricter security requirements who need to have some superusers maintaining the file system but want to have greater control over the z/OS resources that these users can access. Although BPX.DAEMON provides some additional control over the capabilities of a superuser, a superuser should still be regarded as a privileged user because of the full range of privileges the superuser is granted.

The additional control that BPX.DAEMON provides involves the use of kernel services such as `setuid()` that change a caller's z/OS user identity. Any user can issue a `setuid()` which follows a successful `__passwd()` call to the same target user ID. However, a user with daemon authority can issue `setuid()` without knowing the target user's password. With BPX.DAEMON defined, a superuser process can run these types of change services and identity if the following are true:

- The caller's user identity was permitted to BPX.DAEMON.
- All HFS programs running in the address space have been loaded from a library that is controlled by a security product. A library identified to RACF program control is an example. You can identify individual files in the HFS as controlled programs. For more information, "Customizing the system for IBM-supplied daemons" on page 295.

Programs that were loaded from MVS libraries do not need to be controlled programs if the BPX.DAEMON.HFSCCTL FACILITY class profile has been set up. Only HFS files are checked for program control. For information about setting up BPX.DAEMON.HFSCCTL, see "Setting up HFS control" on page 297.

If you do not need to use the default OMVS segment, you can protect privileged users by not allowing them to have OMVS segments. If a service that changes a caller's z/OS user identity, such as `setuid()`, is used, the kernel checks to see if BPX.DAEMON has been defined. If it has, then the kernel checks whether all programs loaded into the address space have been defined to program control. If an uncontrolled program has been loaded, then the address space is marked dirty. In that case, controlled program cannot do any of the controlled functions like `setuid()`. You will see an EMVSERR return code and a JRENVIRTY reason code. See "Handling dirty address spaces" on page 299 for more information about dirty address spaces.

If you set up the default OMVS segment, you need to protect the privileged z/OS users. To do so, define an OMVS segment in their user profile without an UID. This prevents them from picking up the default, and will still protect them from use by a daemon.

However, this additional control does not guarantee that a daemon is well-behaved and only trusts data located in protected sources. An example of this is the **cron** daemon, which uses data that it stores in the HFS to determine how to schedule a work request. If your installation chooses to run a daemon like **cron**, it can protect its privileged z/OS users by not allowing them to have OMVS segments in their security profiles.

Kernel services that change a caller's z/OS user identity require the target z/OS user identity to have an OMVS segment defined. If you want to maintain this extra level of control at your installation, you will have to choose which daemons to permit to the BPX.DAEMON FACILITY class. You will also have to choose the users to whom you give the OMVS security profile segments. To accomplish this, refer to "Steps for preparing the security program for daemons" on page 294.

Steps for preparing the security program for daemons

Before you begin: You need to follow the procedures for security as described in:

1. “Preparing for RACF” on page 56.
2. “Setting up users and groups” on page 60.
3. “Controlling access to files and directories” on page 89.

Perform the following steps to prepare RACF for daemons:

1. Define a BPX.DAEMON FACILITY class profile to permit users that are known as daemons to query or modify the z/OS security environment of a process.

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

Rule: You must use the name BPX.DAEMON. Substitutions are not allowed.

Tip: The system administrator must be defined to the daemon FACILITY class so that if a daemon process fails, the system administrator can restart it. To authorize a current RACF security administrator to be a superuser who can restart daemons, issue:

```
PERMIT BPX.DAEMON CLASS(FACILITY) ID(RACFADM) ACCESS(READ)
```

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

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

-
3. Give daemon authority to the kernel. Most daemons that inherit their identities from the kernel address space are started from **/etc/rc**.

Example: 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
```

-
4. Define a superuser with a user ID of BPXROOT on all systems so that daemon processes can invoke setuid() for superusers.

Example:

```
ADDUSER BPXROOT DFLTGRP(OMVSGRP) OMVS(UID(0))  
          HOME('/') PROGRAM('/bin/sh'))  
NOPASSWORD
```

The NOPASSWORD option indicates that BPXROOT is a protected user ID that cannot be used to enter the system by using a password. The user ID will not be revoked due to invalid logon attempts.

-
5. On the SUPERUSER statement in the BPXPRMxx parmlib member, specify the user ID that the kernel will use when you need a user ID for UID(0).

Example:

```
SUPERUSER(BPXROOT)
```


Rule: If you do not specify the SUPERUSER statement, the default is BPXROOT. In that situation, do not permit the BPXROOT user ID to the BPX.DAEMON FACILITY class profile. The BPXROOT user ID is used when a daemon process invokes `setuid()` to change the UID to 0 and the user name has not been previously identified by `getpwnam()` or by the `_passwd()` function. This action prevents the granting of daemon authority to a superuser who is not defined to the BPX.DAEMON FACILITY class profile.

When you are done, you have prepared RACF for daemons. To complete the security setup, you must also activate program control, as described in “Customizing the system for IBM-supplied daemons.”

Customizing the system for IBM-supplied daemons

z/OS UNIX supplies these daemons:

inetd—the network daemon
rlogind—the remote login daemon
cron—the clock daemon
uucpd—the UUCP daemon

The **syslogd** daemon, which is used to route messages, is shipped with z/OS Communications Services and is documented in their library.

Rules:

- If you are defining BPX.DAEMON for a higher level of security, you need to customize the system for IBM-supplied daemons. Many daemons require BPX.DAEMON authority and must have all modules loaded in their address spaces identified as being defined to program control.
- All modules loaded from LPA are considered to be controlled. RTLS libraries must be defined to program control. If you are using RTLS, you must set up FACILITY profiles as documented in the CSVRTLxx description in *z/OS MVS Initialization and Tuning Reference*.

See *z/OS Security Server RACF Security Administrator's Guide* for information about defining programs to program control.

Defining modules to program control

In most cases, programs loaded into an address space that requires daemon authority must be controlled programs. All HFS programs must be program controlled. However, programs loaded from MVS libraries do not have to be program controlled if the BPX.DAEMON.HFCTL FACILITY class profile has been set up. (See “Setting up HFS control” on page 297.) In that case, only HFS files are checked for program control.

If a program that is not a controlled program is loaded, the address space is marked dirty and cannot perform daemon activities. For more information about dirty address spaces, see “Handling dirty address spaces” on page 299.

Steps for defining programs from load libraries to program control

Before you begin: You need to know which programs you want to define to program control. If you run with enhanced program security, you may need to define some programs with the MAIN attribute via the APPLDATA operand on the PROGRAM profile.

Perform the following steps to define programs from traditional load libraries to program control.

1. Activate the RACF program control (both access control to load modules and program access to data sets).

```
SETROPTS WHEN(PROGRAM)
```

2. Define one of the following profiles.

- a. For a particular program, define a discrete RACF PROGRAM class profile:

```
RDEFINE PROGRAM membername  
  ADDMEM('datasetname'/volser/NOPADCHK) UACC(READ)
```

- b. For all members in a data set:

```
RDEFINE PROGRAM * ADDMEM('datasetname'/volser/NOPADCHK)  
  UACC(READ)
```

3. Refresh the in-storage copy of the PROGRAM profile.

```
SETROPTS WHEN(PROGRAM) REFRESH
```

When you are done, you have defined a program from a load library to program control. For more information about setting up program control for programs from traditional load programs, see *z/OS Security Server RACF Security Administrator's Guide*.

Tips:

1. PROGRAM profile * provides the same function as PROGRAM profile **. If you already have a PROGRAM profile * defined, do not create an ** profile. Instead, issue the RALTER command against PROGRAM * with the same operands shown in the RDEFINE PROGRAM example.
2. If you are running in a sysplex with a shared RACF data base and your system libraries are also shared, then leaving the VOLSER off will allow you to use the same RACF definitions on all systems in the sysplex.
3. Any time you add, change, or delete a profile in the PROGRAM class (with RDEFINE, RALTER, PERMIT, or RDELETE), you must update the in-storage copy of the PROGRAM profile.

```
SETROPTS WHEN(PROGRAM)  
REFRESH
```

4. Daemons that are shipped by z/OS reside in the HFS and are controlled programs, so you do not need to define them to program control. For example, suppose you have a daemon named *server1*. The file **/bin/server1** would have the sticky bit on. Member SERVER1 would reside in SYS1.LINKLIB and be defined as a controlled program.

```
RDEFINE PROGRAM SERVER1
ADDMEM('SYS1.LINKLIB'/'*****'/NOPADCHK) UACC(READ)
SETROPTS WHEN(PROGRAM) REFRESH
```

Tip: You do not need to define the daemons that are shipped by z/OS if you decide to define the BPX.MAINCHECK security profile, as discussed in “Using enhanced program security” on page 300.

5. Daemons can load locales from the HFS or from MVS load modules. If they are loaded from MVS load libraries, then these modules must be marked program-controlled. If they are loaded from the HFS, the program control extended attribute bit must be set. The locales shipped by IBM in the HFS already have this extended attribute bit set.

Step for defining programs in HFS files to program control

Before you begin: The BPX.FILEATTR.PROGCTL FACILITY class controls who can set this flag. You need to have at least READ access before you can set the program control extended attribute.

Perform the following step to set the program control extended attribute.

- Issue the **extattr** command with the **+p** option.

Example: To set the program control extended attribute in the HFS file named *proga*, issue:

```
extattr +p /user/sbin/proga
```

When you are done, you have set the program control extended attribute.

Tip: The attribute is turned off if there is any activity that can change the contents of the HFS file. If this happens, a system programmer with the appropriate privilege will have to verify that the file is still correct. Then the programmer will have to issue the **extattr** command to set the program control attribute back on. To find out if the program control extended attribute has been set, use the **ls -E** command.

Using sanction lists

You can compile a list to contain the lists of pathnames and program names that are sanctioned by the installation for use by program-controlled programs. This file contains properly constructed pathnames and program names as defined in *z/OS UNIX System Services User's Guide*. For more information, see “Using sanction lists” on page 102.

Setting up HFS control

If you want only HFS files to be checked for program control, and do not want programs loaded from MVS libraries to be checked, you can set up the BPX.DAEMON.HFCTL FACILITY class profile. However, doing this weakens some of the security provided by the BPX.DAEMON profile. It should be done only in restricted and carefully considered cases, or if you do not already run with BPX.DAEMON but want to gain only a subset of the benefits of running with BPX.DAEMON.

Steps for setting up HFS control

Before you begin: You need to know whether BPX.DAEMON has already been activated.

Perform the following steps to set up the BPX.DAEMON.HFCTL FACILITY class profile.

1. Activate BPX.DAEMON, if it is not already active. For more information on that topic, see “Steps for preparing the security program for daemons” on page 294.

-
2. Define the resource profile.

```
RDEFINE FACILITY BPX.DAEMON.HFSCTL UACC(NONE)
```

-
3. Give READ access to users.

```
PERMIT BPX.DAEMON.HFSCTL CLASS(FACILITY) ID(uuuuuu) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

When you are done, you have set up the BPX.DAEMON.HFSCTL FACILITY class profile.

Defining HFS files as APF-authorized programs

A program will run APF-authorized if the following requirements are met:

- The program must have been linked with the AC=1 attribute.
- The program must be loaded from an APF-authorized library.
- The program must be the initial program (that is, it must be the job step task program), or it was invoked by a caller that is running APF-authorized.

Rule: If the specified program is going to be invoked as a job step program, you must link-edit it with AC=1. For example:

```
c89 -W1, AC=1
```

In order to avoid possible integrity problems, do not set AC=1 if the program will be run in an APF-authorized environment but not as the job step program (such as DLL).

The APF rules for programs that reside in the HFS are similar to those for programs that reside in authorized libraries. Setting the APF-authorized extended attribute bit should be thought of as putting that program into an authorized library. If you try to run a program from an authorized library that is not linked AC=1, it will not run APF-authorized, but that same program could be fetched by another that is running APF-authorized and executed in the authorization state in which it is called, or even have its state changed.

Tip: To find out whether the APF-authorized extended attribute of the HFS file has been set, use the **Is E** command.

Using sanction lists

You can compile a list to contain the lists of pathnames and program names that are sanctioned by the installation for use by APF-authorized programs. This file contains properly constructed pathnames and program names as defined in *z/OS UNIX System Services User's Guide*. For more information, see “Using sanction lists” on page 102.

Setting the APF-authorized attribute

The BPX.FILEATTR.APF FACILITY class profile controls who can set this flag. You must have the correct permissions before you can set the APF-authorized attribute.

Example: The following example shows the RACF command used to give the necessary permission to user Ralph Smorg with user ID SMORG:

```
RDEFINE FACILITY BPX.FILEATTR.APF UACC(NONE)
PERMIT BPX.FILEATTR.APF CLASS(FACILITY) ID(SMORG) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

Example: In the following example, *proga* is the name of the HFS executable file.

```
extattr +a /user/sbin/proga
```

To set the APF-authorized extended attribute in an HFS executable file, issue the **extattr** command with the **+a** option.

Defining HFS files as shared library programs

A program is loaded as a system shared library program if the HFS executable file has the shared library extended attribute set.

Tip: To find out if the shared library extended attribute has been set, use the **ls -E** command.

Setting the shared library attribute

The BPX.FILEATTR.SHARELIB FACILITY class profile controls who can set the shared library extended attribute. You need to have at least READ access before you can set the shared library extended attribute.

Example: 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.

Example: In the following example, *progdll* is the name of the file.

```
extattr +l /user/sbin/progdll
```

Handling dirty address spaces

Rule: All programs loaded into an address space that requires daemon authority must have been defined to RACF program control. If you have defined the BPX.DAEMON.HFCTL profile in the FACILITY class, then programs that are loaded from MVS libraries are not checked for program control. Only HFS programs are checked for program control. For more information about controlled programs, see *z/OS Security Server RACF Security Administrator's Guide*.

Programs in files are controlled programs if they have the program control attribute set.

If a program that is not a controlled program is loaded, the address space is marked dirty and cannot perform daemon activities. If an address space was marked dirty, you can load a controlled program but it will not be able to do any controlled functions such as `setuid()`. All BPX.SERVER and BPX.DAEMON privileges are revoked, including the right to check passwords.

Programs can be defined to program control in the following ways:

- The load modules can be loaded from a load library, where all modules in the library can be defined to program control, or specific modules in the library can be defined to program control.
- The module can reside in the HFS with the sticky bit on. This causes the system to search the MVS search order and the rules for program control apply as above.
- The module can reside in the HFS with the external attribute set for program control.

RACF supports program control. Other security products may not. If you are using a security product that does not support program control, you may still have BPX.DAEMON defined. In this case, the only thing that will mark an address space dirty is a load from the HFS where the program is not defined to program control.

Dirty address spaces are also known as dirty environments.

Using enhanced program security

If you choose to use the enhanced program security function, and you have daemons that use programs defined to RACF as execute-controlled programs (by having EXECUTE access to the RACF PROGRAM profile that defines the program, or EXECUTE access to the library containing the program), then you need to take some special actions to configure your daemons so that they will run properly.

In an environment with enhanced program security, and using execute-controlled programs, the initial program executed by a daemon must be defined to RACF with a profile in the PROGRAM class, and that profile must specify the MAIN option via the profile's APPLDATA. However, only programs loaded from an MVS library can be defined using the RACF PROGRAM class; you cannot define programs loaded from the z/OS UNIX file system. Therefore, if you have daemons that use execute-controlled programs, you need to move their initial program from the z/OS UNIX file system into an MVS library so that you can define it completely to RACF.

Additionally, if you run with enhanced program security and have FACILITY profile BPX.DAEMON defined, you can use another FACILITY profile to request that z/OS UNIX apply tighter security controls to your daemons. Normally, with BPX.DAEMON defined, z/OS UNIX will work with RACF to enforce a clean execution environment for any daemon. In this case, the daemon can run only those programs defined to the RACF PROGRAM class or marked controlled via the **extattr** shell command with the **+p** option.

For additional security, you can define FACILITY profile BPX.MAINCHECK. When you do that, z/OS UNIX and RACF will require that the first program your daemon executes must be defined to RACF using a PROGRAM profile with the MAIN option, as described above for use of execute-controlled programs. If you define BPX.MAINCHECK, then you need to move the first program that any daemon executes from to an MVS library if it currently resides in the UNIX file system.

Steps for setting up enhanced program security

Before you begin: You need to have:

1. RACF set up as your security product
2. Enabled RACF enhanced program security
3. Enabled BPX.MAINCHECK

4. determined which privileged HFS programs you run that are affected by setting up RACF enhanced program security. The RACF programs that would be affected are the main jobstep programs of one of the following types of privileged applications:
 - z/OS UNIX applications that require a program controlled environment. This includes applications that require permission to BPX.DAEMON, BPX.SERVER or BPX.SRV.userid or those that use a privileged function like `__passwd()`. Examples of applications that would be affected by this are `rlogin`, `telnet` and `su`.
 - Applications that gain access to MVS data sets by using RACF program access to data sets (PADS) via entries in a DATASET profile's conditional access list.

Perform the following steps to set up enhanced program security.

1. Turn on RACF enhanced security checking. For more information, see *z/OS Security Server RACF Security Administrator's Guide*.

2. Ensure that all affected HFS MAIN jobstep programs are in an MVS load library in your MVS load library search order. They should have either the sticky bit HFS attribute turned on (see "Steps for verifying that the sticky bit is on" on page 312) or have been set up as an external link HFS file (see "Step for using external links to access MVS load libraries" on page 313).

If you use the warning mode provided by RACF enhanced program security as a way to determine which programs will be affected by the new enhanced security checking, note that in warning mode, the applications will not fail but you will get messages that indicate which programs are affected

3. Define the BPX.MAINCHECK security profile.

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

4. Re-IPL.

When you are done, you have set up enhanced program security.

Tips:

1. You can partially activate enhanced program security by defining the profile before restarting OMVS or issuing a SET OMVS or SETOMVS command. However, only address spaces that are started after enhanced program security was enabled are affected. Use this partial enablement for testing purposes only.
2. Because the new RACF enhanced security checking requires a completely controlled program environment, testing using **dbx** may be restricted because it can cause the program environment to be considered uncontrolled. Testing a trusted MAIN program under **dbx** may require that the RACF enhanced security checking be set up in warning mode or that BPX.MAINCHECK be undefined. Attempting to do otherwise may cause some privileged operations to fail while under **dbx** control.

Recommendation: You should remain in warning mode until you have done at least one IPL, to ensure that you have tested with all your daemons.

Customizing the system for IP-supplied daemons

The **syslogd** daemon, which is used to route messages, is shipped with z/OS Communications Services (TCP/IP Services). Other daemons provided by z/OS Communications Services are **otelnetd** and **orexecd**.

Rule: Before you can use the daemons, you have to permit each daemon to the BPX.DAEMON FACILITY profile and then ensure that the library that contains the daemon is added to the program control profile.

Steps for customizing the system for IP-supplied daemons

Before you begin: You must know what IP-supplied daemons you will be using.

Perform the following steps to customize the system for IP-supplied daemons.

1. Permit each daemon to the BPX.DAEMON FACILITY class profile.

Example: Set up the **syslogd** daemon, which is in the SEZALOAD library:

```
PERMIT BPX.DAEMON CLASS(FACILITY) ID(syslogd) ACCESS(READ)
```

2. Add the library that contains the library to the program control profile.

Example: Define the SEZALOAD library to the PROGAM class:

```
RALT PROGRAM * ADDMEM('tcpip.SEZALOAD'/'volser'/NOPADCHK) UACC(READ)
```

Result: You have set up the **syslogd** daemon and then added the SEZALOAD library to the program control profile. You can start using that daemon.

See “Customizing the system for IBM-supplied daemons” on page 295 for more information on program control. *z/OS Communications Server: IP Configuration Guide* also has more information.

When you are done, you have customized the system for IP-supplied daemons.

Customizing the IBM-supplied daemons

This section discusses customizing these IBM-supplied daemons: **inetd**, **uucpd**, **rlogind**, and **cron**.

Customizing the inetd daemon

The **inetd** daemon provides service management for a network. It reduces system load by invoking other daemons only when they are needed and by providing several simple Internet services internally without invoking other daemons.

After it has been started, the **inetd** daemon monitors network sockets for services listed in **/etc/inetd.conf**. That file tells the **inetd** daemon which services to support and how to handle service requests. When **inetd** receives a request on one of these sockets, it determines which service corresponds to that socket. Then it either handles the service request itself or invokes the appropriate server. For more information about the **/etc/inetd.conf** file, see the description of **inetd** in *z/OS UNIX System Services Command Reference*.

For z/OS UNIX, **inetd** handles rlogin, telnet, rsch, rexec, and others. It uses a configuration file in **/etc/inetd.conf** when handling the requests.

Steps for customizing the inetd daemon

Before you begin: TCP/IP must be properly configured and started.

Perform the following steps to customize the inetd daemon.

1. Copy **/samples/inetd.conf** to **/etc/inetd.conf**.

2. Decide which services you want to support, such as rlogin and telnet. There is no list of daemons that can be started from **inetd**; any daemon that can run under **inetd** should say so in its documentation and also tell you what its **inetd.conf** entry should look like.

3. Decide on a user name for the services. You can use the one in the sample **inetd.conf** (OMVSKERN). You can also use a different user name for each service. Some daemons may not require as many privileges as others.

4. Set up the user names in RACF, with appropriate privileges. You should consider whether to use BPX.DAEMON support. (For more information, see “Establishing the correct level of security for daemons” on page 291.)

5. Uncomment or add a line in **inetd.conf** for each service that you want to support. Make any changes needed to the lines for supported services. See the description of **inetd** in *z/OS UNIX System Services Command Reference* for the syntax of **inetd.conf** entries. Also refer to the appropriate documentation for the various daemon programs for the requirements for each daemon.

6. Make sure that each service is listed in TCPIP.ETC.SERVICES or **/etc/services** with the appropriate port number.

7. Arrange for **inetd** to be started on each IPL. The most common way to do this is to start it from **/etc/rc** but it can also be started from a started task using BPXBATCH with PARM='SH...' or from a shell session of a user with appropriate authority.

When you are done, you have customized the **inetd** daemon.

Customizing the uucpd daemon

The **uucpd** daemon handles the communications between local and remote sites for file transfer via TCP/IP connections in an UUCP network. For more information, see Chapter 12 and the descriptions of the various **uucp** commands in *z/OS UNIX System Services Command Reference*.

Steps for customizing the uucpd daemon

Before you begin: You need to set up the **uucpd** daemon to **/etc/inetd.conf**.

Example: Add the following lines to **/etc/inetd.conf**:

```
uucp stream tcp nowait omvskern /usr/sbin/uucpd
uucpd -l0
```

If you want to have the **uucpd** daemon run with a user ID other than OMVSKERN (for example, UUCPD), you need to decide what the new user ID will be.

Perform the following steps to customize the **uucpd** daemon.

1. Change the line in **/etc/inetd.conf** to:

```
uucp stream tcp nowait uucpd /usr/sbin/uucpd uucpd -l0
```

2. Define user ID UUCPD to RACF.

```
ADDUSER UUCPD DFLTGRP(OMVSGRP)
OMVS(UID(0) HOME('/') PROGRAM('/bin/sh'))
NOPASSWORD
```

The NOPASSWORD option indicates that this is a protected user ID that cannot be used to enter the system by using a password. The user ID will not be revoked due to invalid logon attempts. In this case, you are defining the UUCPD user ID without a TSO/E segment.

When you are done, you have customized the **uucpd** daemon so that it runs with the UUCP user ID.

Customizing the rlogind daemon

The **rlogind** daemon validates rlogin requests. If you choose to have the **rlogind** daemon run with a user ID other than OMVSKERN, do the following:

Step for customizing the rlogind daemon

Before you begin: You need to know what user ID you want the **rlogind** daemon to run with.

Perform the following step to customize the **rlogind** daemon so that it runs with a user ID other than OMVSKERN (for example, RLOGIND).

- Issue the following command:

```
ADDUSER RLOGIND
DFLTGRP(OMVSGRP) OMVS(UID(0) HOME('/')
PROGRAM('/bin/sh')) NOPASSWORD
```

The NOPASSWORD option indicates that this is a protected user ID that cannot be used to enter the system by using a password. The user ID will not be revoked due to invalid logon attempts.

In this case, you are defining the user ID without a TSO/E segment.

When you are done, you have customized the **rlogind** daemon so that it runs with a user ID other than OMVSKERN.

Tips:

- For all the other setup steps required for **rlogin**, see “Setting up for rlogin” on page 317.
- If you are writing or porting your own command to process login requests, the shell interface to rlogin is the FOMTLINP module, documented in Appendix B. FOMTLINP has many parameters that can be used to tailor the rlogin processing. FOMTLINP is the login function and FOMTLOUT is the logout function. For more information about the **rlogin** daemon, see *z/OS UNIX System Services Command Reference*.

Customizing the cron daemon

cron is a clock daemon that runs commands at specified dates and times. The **/usr/lib/cron/at.allow** and **/usr/lib/cron/at.deny** files control access to the **at** and **batch** commands. The **/usr/lib/cron/cron.allow** and **/usr/lib/cron/cron.deny** files control access to **crontab**. These are text files that contain user IDs. See *z/OS UNIX System Services Command Reference* for information about the **cron** daemon and controlling access.

Because **cron** may attempt to send mail, you should also configure the **mailx** utility as described in “Customizing electronic mail” on page 211.

Starting and restarting daemons

Daemons can be started by JCL and also by the shell. Some daemons such as **inetd** can also be started by the shell. Interactive login shells, shell scripts run as background jobs from a login shell, and batch jobs using BPXBATCH to run the shell all can start daemons.

BPXBATSL is provided as an alias for BPXBATCH. BPXBATSL performs a local spawn but does not require the resetting of environment variables. BPXBATSL behaves exactly like BPXBATCH, and allows local spawning whether the current environment is set up or not.

Starting a daemon from the shell

Many daemons can be started from the shell, both interactively and from shell scripts. In general, processes started from the shell complete (either successfully or with some error) before the parent shell itself exits. Any processes still running receive a SIGHUP signal when the parent shell exits. The default action for SIGHUP is to terminate the process. That is, when the shell exits, the system terminates all running processes started by the shell.

Daemon processes are long-running and generally must continue to run even after the invoking shell terminates. Those daemons started using the shell are therefore written to ignore SIGHUP signals. They are also typically written to return control to the shell immediately. If they did not return, the shell script would wait forever for the daemon to exit.

Rules:

- When started from the shell, most daemons should not be placed in the background environment. That is, an ampersand should not appear on the shell command line that starts a daemon. Doing so would expose the background job containing the daemon to SIGHUP and cause the daemon to terminate unexpectedly when the shell script exits.
- Some daemons either do not protect themselves from the SIGHUP signal or do not return to the shell immediately. You have to have those daemons start in the

background environment. To do this, add an ampersand character at the end of the command line that starts the daemon. (The **syslogd** daemon does not need the &.)

- When starting daemons in the background environment, it is very important to include a **sleep** command at the end of the script. This command gives the background processes time to get started and set up to ignore SIGHUP so that when the shell exits, the daemons keep running when the shell script completes. The amount of time required can be determined empirically. A value of 5 seconds is suggested for a start.

A shell script that starts a more simple daemon called **slowpoke** that does not return control immediately to the shell would look like this:

```
slowpoke &  
sleep 5  
exit
```

In summary, a shell script that starts the **syslogd** daemon would look like the following:

```
_BPX_JOBNAME='SYSLOGD' /usr/sbin/syslogd -f /etc/syslog.conf &  
_BPX_JOBNAME='CROND' /usr/sbin/cron &  
sleep 5  
exit
```

Although **cron** and **syslogd** return immediately and protect themselves from SIGHUP, the & is included with **syslogd** because this is the only method of getting **_BPX_JOBNAME** to take effect.

Using & at the end of a command

Using an & at the end of a command starts the command in the background. The shell forks a child process, executes the command program, and then does not wait for the command to complete. Some daemons must be started this way in order to allow the invoking shell script (such as **/etc/rc**) to continue. **cron** does not need to be started with an & because it forks itself to create the child process, which continues running while the **cron** parent process returns to the invoker such as **/etc/rc**. If the script does **/usr/sbin/cron**, the shell will spawn the **cron** program to create a child process, and then the **cron** program will fork a child process to run the daemon independently. The **cron** command returns to the shell, and the script continues.

However, system programmers may want the **cron** daemon process to have a jobname. To do this from a shell script, you can use the **_BPX_JOBNAME** environment variable. (This can be done on the command line, or in a prior export command.) The **_BPX_JOBNAME** variable assigns the jobname to exec'd programs, running in forked processes, but not to locally spawned processes. As a result, the shell command

```
_BPX_JOBNAME=CROND /usr/sbin/cron
```

may not assign the jobname to the **cron** daemon. (It depends if the spawn is done within the same address space.) But, the shell command

```
_BPX_JOBNAME=CROND /usr/sbin/cron &
```

will assign the jobname to the **cron** daemon, because it is run with a fork/exec.

Starting and restarting daemons

There are several ways to start and restart daemons. The method used depends on the level of control the installation has chosen for daemons. You can start daemons in any of these ways:

- Put the command in **/etc/rc** to start the daemon automatically during initialization. For information on starting programs from **/etc/rc**, see “Customizing /etc/rc” on page 201.

When UNIX systems are initialized (IPLed or restarted), the **/etc/rc** shell script is run to perform system initialization functions and to start daemons. If a daemon terminates, a superuser must restart the daemon. Another alternative on many UNIX systems is a function called **inittab**, which is a program that processes a table. The table indicates which daemons to start and what to do if the daemon stops. However, z/OS UNIX does not provide support for **inittab**.

The following explanation uses the **syslogd** daemon (which supplies logging functions for programs) as an example of a daemon. Similar steps are required for other daemons.

syslogd is typically started from **/etc/rc**, as shown in the following example:

```
_BPX_JOBNAME='SYSLOGD' /usr/sbin/syslogd -f /etc/syslog.conf
```

In this example, the **_BPX_JOBNAME** environment variable is set to assign a job name of **SYSLOGD** to the **syslogd** daemon. This allows the operator to have better control over managing the **syslogd** daemon.

When started from **/etc/rc**, stdin and stdout are set to **/dev/null** and stderr is set to **/etc/log** for recording any errors. If the **syslogd** process fails, you could re-IPL, but this would be very disruptive to the users.

- You can also start **syslogd** with a cataloged procedure, as shown in the following example:

```
//SYSLOGD  PROC
//SYSLOGD  EXEC PGM=SYSLOGD,REGION=30M,TIME=NOLIMIT
//          PARM='POSIX(ON) ALL31(ON)/ -f /etc/syslog.conf'
//SYSPRINT  DD SYSOUT=*
//SYSIN     DD DUMMY
//SYSOUT    DD SYSOUT=*
//SYSERR    DD SYSOUT=*
//CEEDUMP   DD SYSOUT=*
```

For this **syslogd** cataloged procedure to get control with superuser and daemon authority, you must add an entry to the started procedures table, or define it in the **STARTED** class.

It is suggested that you assign user ID **OMVSKERN** to **SYSLOGD** in the RACF started procedures table. For example:

```
DC  CL8'SYSLOGD'  PROCEDURE NAME
DC  CL8'OMVSKERN' USER ID (to be used for SYSLOGD proc)
DC  CL8'OMVSGRP'  GROUP NAME OR BLANKS FOR USER'S DEFAULT GROUP
DC  XL1'00'       NOT TRUSTED
DC  XL7'00'       RESERVED
```

For more information about the Started Procedures table, see *z/OS Security Server RACF System Programmer's Guide*.

To start **syslogd**, issue the following command from the console:

```
S SYSLOGD
```

Whenever the **syslogd** daemon is deactivated, you can issue this command to restart it.

- Another method is with a cataloged procedure using BPXBATCH to invoke a daemon program located in the HFS.

```
//SYSLOGD PROC
//SYSLOGD EXEC PGM=BPXBATCH,REGION=30M,TIME=NOLIMIT,
//      PARM='PGM /usr/lpp/tcpip/sbin/syslogd -f /etc/syslogd.conf'
//* STDIN and STDOUT are both defaulted to /dev/null
//STDERR DD PATH='/etc/log',PATHOPTS=(OWRONLY,OCREAT,OAPPEND),
//      PATHMODE=(SIRUSR,SIWUSR,SIRGRP,SIWGRP)
```

syslogd requires superuser and daemon authority.

The JCL in the SYSLOGD PROC invokes BPXBATCH, which sets up the standard file descriptors and environment variables before running **/usr/lpp/tcpip/sbin/syslogd**.

Tip: In order to reference the **syslogd** messages from the message catalog files in the HFS, you must create a symbolic link to the **syslogd.cat** file. With a superuser ID, in one of the z/OS UNIX shells, issue the following command:

```
ln -s /usr/lpp/tcpip/lib/nls/msg/C/syslogd.cat
    /usr/lib/nls/msg/C/syslogd.cat
```

For more information about the syslog daemon, see *z/OS Communications Server: IP Configuration Guide*.

Setting up security procedures for daemons

Recommendation: You should consider setting up security for daemons if you plan to take advantage of z/OS UNIX.

Steps for setting up security procedures for daemons

Before you begin: You need to assume the following:

- You want the added system integrity of having BPX.DAEMON defined.
- Daemons will share the OMVSKERN user ID and be started from **/etc/rc**.

Perform the following steps to define and start daemons.

1. Define the group OMVSGRP.

```
ADDGROUP
OMVSGRP OMVS(GID(1))
```

2. Define the user OMVSKERN.

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

NOPASSWORD indicates that OMVSKERN is a protected user ID; it cannot be used to enter the system by using a password. The user ID will not be revoked due to invalid logon attempts.

3. Add the daemon cataloged procedure to the RACF STARTED class or the Started Procedure table, module ICHRIN03. Do not make it trusted. See “Preparing RACF” on page 56.

-
4. Create the BPX.DAEMON FACILITY class profile.

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

-
5. Grant daemon authority to the kernel.

```
PERMIT BPX.DAEMON  
CLASS(FACILITY) ID(OMVSKERN) ACCESS(READ)
```

-
6. Activate program control if you have not already done so and ensure that the daemon programs and Language Environment run-time library are in a library that is controlled by z/OS.

```
SETOPTS WHEN(PROGRAM)  
RDEFINE PROGRAM * ADDMEM  
( 'CEE.SCEERUN' /RTLPAK/NOPADCHK  
'SYS1.LINKLIB' /'*****' /NOPADCHK) UACC(READ)  
SETOPTS WHEN(PROGRAM) REFRESH
```

Change RTLPAK to the pack that the PDS resides on.

Tip: You can use PROGRAM PROFILE ** instead of PROGRAM PROFILE *.

When you are done, you have set up and defined daemons.

Giving daemon authority to vendor-written programs

If you are writing a program that uses z/OS UNIX services to change user identity (such as using `setuid` and `seteuid` and so forth), you should refer to “Setting up the BPX.* FACILITY class profiles” on page 83 to determine whether your program’s use of these services requires it to have DAEMON authority.

If you determine that your program requires DAEMON authority, then you need to do the following:

1. Document the requirement to assign a user ID to the daemon which has an UID of 0.
2. Document the requirement to permit the daemon to the BPX.DAEMON FACILITY class profile.
3. Document how to start the daemon from **/etc/rc** or as a started procedure.
4. The main program and all programs that it loads must be marked program controlled in the HFS or be loaded from an MVS program controlled library. For more information on marking programs program-controlled in the HFS, see “Customizing the system for IBM-supplied daemons” on page 295. For more information on placing your HFS programs into MVS libraries, see “Moving HFS executables into the link pack area” on page 336. This section describes steps to move a program into LPA; similar steps can be followed to move a program

into a system linklist library or a step library. If you decide to use MVS libraries, you need to also see “Steps for defining programs from load libraries to program control” on page 296.

Tracking down problems when setting up daemons and servers

This section describes possible problems you may encounter when setting up daemons and servers. It covers the following topics:

Problem	Reference
A user or daemon is not properly defined with an OMVS segment.	“Verifying the user OMVS segment”
A group is not properly defined with an OMVS segment.	“Verifying the group OMVS segment” on page 311
A module that is not defined to program control was loaded into the daemon’s address space.	“Using external links to access MVS load libraries” on page 313
The daemon module is coming from the HFS. The sticky bit may be off or the program may not be in the MVS search order.	“Verifying that the sticky bit is on” on page 312 and “Using external links to access MVS load libraries” on page 313
The executable file being loaded from the file system does not have the program control extended attribute set.	“Finding modules that were not defined to program control” on page 313
The file system containing a program control executable file was mounted with the NOSETUID option. This makes the entire HFS uncontrolled.	“Finding modules that were not defined to program control” on page 313
The daemon does not have READ or higher permission to the BPX.DAEMON.FACILITY.	“Checking the daemon authority” on page 315
The server does not have READ or higher permission to BPX.SERVER.FACILITY.	“Checking the server setup” on page 315
The in-storage data that is managed by RACF may be out of date.	“Refreshing RACF in-storage data” on page 316
The server may not have been defined to the BPX.SRV.aaaaaaaa SURROGAT class profile.	“Checking the SURROGAT class profile” on page 316

Verifying the user OMVS segment

If a user or daemon is not properly defined with an OMVS segment, you need to verify that the user or daemon has an OMVS segment with an UID.

Step for verifying the user OMVS segment

Perform the following step to verify the user OMVS segment.

- Issue the LU command.

Example: For the DAEMONU daemon user ID, issue:

```
LU DAEMONU OMVS
```

Result: You will see output similar to the following:


```

LU DAEMONU OMVS

USER=DAEMONU NAME=UNKNOWN OWNER=WELLIE CREATED=92.104
DEFAULT-GROUP=DAEMONG PASSDATE=92.125 PASS-INTERVAL=N/A
ATTRIBUTES=SPECIAL OPERATIONS
...
GROUP=DAEMONG AUTH=USE CONNECT-OWNER=WELLIE CONNECT-DATE=92.104
CONNECTS= 82 UACC=NONE LAST-CONNECT=95.261/14:09:38
...
OMVS INFORMATION
-----
UID= 0000000000
HOME= /
PROGRAM= /bin/sh
CPUTIMEMAX=NONE
ASSIZEMAX=NONE
PROCUSERMAX=NONE
THREADSMAX=NONE
MMAPAREAMAX=NONE

```

You should now see that the UID is 0. (The UID for all daemons must be 0, which gives superuser authority to the daemon.)

Verifying the group OMVS segment

If a group is not properly defined with an OMVS segment, you need to verify that the user has a group OMVS segment with a GID defined.

Step for verifying the group OMVS segment

Perform the following step to verify the group OMVS segment.

- Issue the LG command.

Example:

```
LG DAEMONG OMVS
```

Result: You will see output similar to the following:

```

LG DAEMONG OMVS
INFORMATION FOR GROUP DAEMONG
SUPERIOR GROUP=NONE OWNER=IBMUSER
...
USER(S)= ACCESS= ACCESS COUNT= UNIVERSAL ACCESS=
DAEMONU JOIN 000392 READ
...

OMVS INFORMATION
-----
GID= 0000000500

```

When you are done, you can check the UID. For this example, it is 500.
(Installations can choose the GID values for their groups.)

Verifying that the sticky bit is on

If the daemon module resides in an MVS load library, the file containing the daemon module must have the sticky bit set on.

Steps for verifying that the sticky bit is on

Before you begin: You must know which shell you are using.

Perform the following step to verify that the sticky bit is on.

- Base your action on the shell that you are using.

If you are using this. . .	Then . . .
The ISPF shell	From the ISPF shell, enter the file name of the daemon (/usr/sbin/daemon1, for example) and request Attributes. Result: You should see a display similar to the following: Display File Attributes Pathname : /usr/sbin/daemon1 Link count : 2 Set UID bit : 0 Set GID bit : 0 Sticky bit : 1 ... In the line for the sticky bit, 1 indicates that the sticky bit is on
The z/OS UNIX shell	Issue ls -l Result: You will see a display similar to the following: -rwxr--r-T 2 SUPERU SYS2 131072 Oct 25 10:19 daemon1 T indicates that the sticky bit for daemon1 is on.

When you are done, you have checked whether the sticky bit is on.

Rules:

1. If the daemon module resides in the HFS, the file containing the daemon module must have the program control extended attribute set.
2. If the HFS program does have the extended attribute set, you still need to verify that the HFS that it resides in is not mounted with the NOSETUID option. Do one of the following:
 - Check the MOUNT statement in BPXPRMxx.
 - Display the file system information by using the **df** command. The file system, the mount table, and ISHELL have attributes that enable you to see this setting:

File system name:
OMVS.HFS.SMORG
Mount point:

```
/u/smorg  
  
Blocks in use . . . . : 72403  
Ignore SETUID . . . . : 1
```

If the “Ignore SETUID” value is set to 1, loading modules from this HFS will mark your address space dirty. For more information about dirty address spaces, see “Handling dirty address spaces” on page 299.

Using external links to access MVS load libraries

Instead of using the sticky bit for programs that are invoked via either `exec()` or `spawn()`, or are loaded with the `dllload()` function, use an external link to an MVS program name. Both functions search the MVS load library search order for the MVS program.

Tip: If you use an external link, the MVS program name defined by the external link does not have to be part of the file name of the HFS program.

Step for using external links to access MVS load libraries

Perform the following step to use external links to access MVS load libraries.

- Define an HFS file as an MVS program name in an external link.

Example: Define an HFS file `/usr/lpp/internet/bin/wwwss.so` as MVS program name IMWYWWS in an external link.

```
ln -e IMWYWWS /usr/lpp/internet/bin/wwwss.so
```

When you are done, you have created an external link that can be used to access an MVS load library.

Finding modules that were not defined to program control

If an HFS module that was not defined to program control is loaded into an address space and a process in the address space tries to invoke a restricted z/OS UNIX service such as `setuid()`, you get the JRENVDIRTY reason code. It indicates a dirty address space; see “Handling dirty address spaces” on page 299 for more information.

Steps for finding modules that were not defined to program control

Before you begin: You need to check your job log and have the security administrator check the security console for diagnostic messages.

Perform the following steps to find the module that was not defined to program control.

1. Search the RACF database for a list of the modules that are defined to program control.

Example: Issue the following TSO/E command:

```
SEARCH CLASS(PROGRAM) NOMASK
```

Result: You will see output similar to the following:

```
CEE0LVD  
CEE0V  
CEEPLPKA  
CEEZ24  
DAEMON
```

```
EDCUCSNM
EDCUEYI1
EDC$EUEY
...
```

2. Look for the daemon module (for example, DAEMON) and locations in the format EDC\$xyxy (in the output in Step 1, EDC\$EUEY is the module for the US English locale).

3. If the output of the SEARCH module shows *, issue:

```
RLIST PROGRAM *
```

The * will cover any module name in the libraries displayed in the output of the RLIST command. If a VOLSER is displayed with a library name, make sure that the VOLSER is also correct.

4. Gather data about which programs need to be defined to program control by using SLIP. The complete details are in *z/OS Security Server RACF Diagnosis Guide*.

Example:

```
SLIP SET,IF,ACTION=TRACE,LPAMOD=(ICHRFR00,xxxxx),J=jobname,
TRDATA=(STD,REGS,zzzzzz),ML=100,END
```

5. Because this SLIP produces GTF records, you must start GTF. Be sure that you specify PARM TRACE=SLIP. Then use IPCS to format the data with the GTFTRACE IPCS command. You will see output similar to the one in Figure 50.

SLIP S+U	ASCB.... 00FAF580 CPU..... 0001	JOBN.... INETD8
		
	GENERAL PURPOSE REGISTER VALUES	
	0-3..... 7FFEB744 7FFEB748 00000000 007F2978	
	4-7..... 0000000C 007F0738 00000004 007F24D8	
	8-11.... 00000000 7FFEB6A8 80E2323E 007F2978	
	12-15... 00000000 7FFEB6A8 80E23616 0000000C	
	...	
SLIP USR	CPU..... 0001 EXT..... 0001	CNTLN... 00
0008	C3C5C5C2 C9D5C9E3	CEEBINIT
002C	C3C5C54B E2C3C5C5 D9E4D540 40404040	CEE.SCEERUN
	40404040 40404040 40404040 40404040	
	40404040 40404040 40404040	
0006	D6D7F2D9 E2F1	OP2RS1

Figure 50. GTFTRACE output for tracking modules for program control

6. Look for a SLIP S+U entry where R15 has a value of 0000000C. Then look at the SLIP USR entry to identify the module and library that needs to be defined to program control.

You know you are done when you have identified the module and library that needs to be defined to program control.

Tip: To define the module to program control, issue:

```
RDEFINE PROGRAM CEEBINIT ADDMEM -  
( 'CEE.SCEERUN' /OP2RS1/NOPADCHK) UACC(READ)
```

Checking the daemon authority

You may need to check to see if the daemon has READ or higher permission to the BPX.DAEMON FACILITY class profile.

Step for checking the daemon authority

Perform the following step to see if the daemon has READ or higher permission to the BPX.DAEMON FACILITY class profile.

- Issue the following command:

```
RLIST FACILITY BPX.DAEMON AUTHUSER
```

You will see output similar to the one in Figure 51.

```
RLIST FACILITY BPX.DAEMON AUTHUSER  
  
CLASS      NAME  
-----  
FACILITY   BPX.DAEMON  
...  
USER       ACCESS  ACCESS COUNT  
-----  
...  
DAEMONU    UPDATE   000007  
...
```

Figure 51. Output from command `RLIST FACILITY BPX.DAEMON AUTHUSER`

You should now see that the daemon (DAEMONU in this example) has UPDATE permission to the BPX.DAEMON FACILITY class profile.

Checking the server setup

You can have similar problems setting up a server as when setting up daemons. All of the steps for verifying program control apply to servers as well as daemons, but instead of checking the BPX.DAEMON FACILITY class profiles, verify that the BPX.SERVER profile is properly defined.

Step for checking the server authority

Perform the following step to check if the BPX.SERVER profile has been properly defined with READ or higher permission.

- Issue the following command:

```
RLIST FACILITY BPX.SERVER AUTHUSER
```

You will see output similar to the one in Figure 52 on page 316.

```
RLIST FACILITY BPX.SERVER AUTHUSER
```

CLASS	NAME		
----	----		
FACILITY	BPX.SERVER		
...			
USER	ACCESS	ACCESS	COUNT
----	----	----	----
...			
SERVERU	UPDATE		000007
...			

Figure 52. Output from command `RLIST FACILITY BPX.SERVER AUTHUSER`

You should now see that the BPX.SERVER profile has been properly defined with READ or higher permission.

Refreshing RACF in-storage data

RACF uses the RACLIST option of the SETROPTS command to define what profile information is to be buffered in storage for faster performance. If you think that you have defined everything correctly, try refreshing the various profiles relating to the daemon and server support.

Following is the set of all relevant refresh commands:

```
SETROPTS WHEN(PROGRAM) REFRESH
SETROPTS RACLIST(FACILITY) REFRESH
SETROPTS RACLIST(SURROGAT) REFRESH
```

Checking the SURROGAT class profile

If your server processes user requests without a password, it must be defined to a SURROGAT class profile for the user ID.

Step for checking the SURROGAT class profile: Perform the following step to check whether the server has been defined to the BPX.SRV.aaaaaaaa SURROGAT class profile.

- Issue the appropriate RLIST command.

Example: Assuming that your server needs to process requests from user ID ANONYMOS, issue:

```
RLIST SURROGAT BPX.SRV.ANONYMOS AUTHUSER
```

Result: You will get output similar to the one in Figure 53.

```
RLIST SURROGAT BPX.SRV.ANONYMOS AUTHUSER
```

CLASS	NAME		
----	----		
SURROGAT	BPX.SRV.ANONYMOS		
...			
USER	ACCESS	ACCESS	COUNT
----	----	----	----
...			
SERVERU	READ		000007
...			

Figure 53. Output from command `RLIST SURROGAT BPX.SRV.ANONYMOS AUTHUSER`

When you are done, you can verify that the user ID that you are running your server on (in this example, SERVERU) has READ or higher permission to the BPX.SRV.userid SURROGAT class profile.

Setting up for rlogin

You can use **rlogin** to log on to a z/OS UNIX system from a remote system. Two daemons are used when processing rlogin requests:

- The **inetd** daemon handles rlogin requests.
- The **rlogind** daemon is the server that validates the remote login request and checks the password. It does not have a customization file in the HFS.

The z/OS UNIX system does not use the **.rhosts** file that many UNIX systems use. It indicates the remote hosts and users who can access your system without specifying a password. A password is always required to rlogin to a z/OS UNIX system.

Steps for setting up for rlogin

Before you begin:

1. You need to set up the appropriate security as described in:
 - a. “Preparing for RACF” on page 56.
 - b. “Setting up users and groups” on page 60.
 - c. “Controlling access to files and directories” on page 89.
 - d. “Setting up TCP/IP security” on page 108
2. You also need to customize the SUBFILESYSTPE statement in the BPXPRMxx parmlib member to include TCP/IP. For more information, see *z/OS MVS Initialization and Tuning Reference*.

Perform the following steps to set up for rlogin.

1. Set up security for the **inetd** and **rlogind** daemons. See “Steps for preparing the security program for daemons” on page 294.
2. Establish the connection between TCP/IP and z/OS UNIX; refer to Chapter 21.
 - a. Define port 513 in **/etc/services** (if this file is to be used) or in the hlq.ETC.SERVICES data set, where *hlq* is the prefix defined by DATASETPREFIX in the TCP/IP profile (“TCPIP” by default). If **/etc/services** is defined, it will be used instead of hlq.ETC.SERVICES. The format of the line is:

```
login 513/tcp
```
 - b. IPL or re-IPL if needed.
 - c. Start TCP/IP.
3. Customize **/etc/inetd.conf**. It tells **inetd** how to handle Internet service requests on Internet sockets. If you do not have that file, copy it from **/samples**.

Example:

```
cp /samples/inetd.conf /etc/inetd.conf
```

Make these changes in the file:

- a. Change the user ID of the login server (which is **rlogind**) to an ID with daemon authority.
`login stream tcp nowait OMVSKERN /usr/sbin/rlogind rlogind -m`
- b. Comment out any servers that are not needed by putting a # in the first column.

-
4. Start the **inetd** daemon. It is normally started from **/etc/rc**, which is executed when the system is initialized. Put these lines in **/etc/rc**, or uncomment them out, as the case may be:

```
_BPX_JOBNAME='INETD' /usr/sbin/inetd /etc/inetd.conf&
```

Tips:

- When you start **inetd** from the shell, you need to do it from an OMVS session.
 - If TCP/IP is not yet up, you will receive error messages, but **inetd** will try again in a few minutes.
 - To obtain debugging information, issue:
`/usr/sbin/inetd -d`
 - To verify that **inetd** is listening on port 513, issue the TSO NETSTAT INTERVAL command and check the output.
-

When you are done, you can issue the **rlogin** command to log in from a remote UNIX system.

Problem determination

This section describes possible problems you may encounter when setting up rlogin. You can use any of these to aid in problem determination:

- The **-d** option of the **inetd** command
- The **-d** option of the **rlogind** command
- The **/tmp/.stderr** file

If there are problems on the client side, you may get the following messages:

invalid logon name or password

This message is misleading and often is caused by setup problems on the z/OS UNIX side. It is possible that security was not set up correctly.

If there are problems on the server side, you may get the following messages:

Resource temporarily unavailable

inetd will try initializing the service every three minutes.

service unavailable

This usually means that the port assignment is incorrect. Use the TSO NETSTAT INTERVAL command to verify that OMVS issued a LISTEN for port 513. If 513 is not there, then **inetd** cannot find the port assignment for 513 in **/etc/services** or **hlq.ETC.SERVICES**. See Step 2 on page 317.

Chapter 18. Preparing security for servers

Overview

This chapter describes security for your server applications. It uses the word “server” to mean “server application”, which is an application that provides a service for clients. This server could be part of a software product that will run on any company’s z/OS computing environment, or it might be written by your application programmers for your own company’s use. This chapter is intended for both audiences:

- The application programmers designing the server. They must decide what kind of security the server is to have so they can code for it and provide documentation (either verbally or in writing) for those who will run the server.
- The security administrator at the company that runs the server. They must set up the profiles based on the documentation provided with the server.

It is important that both audiences read all the topics in this chapter. This means, for example, that security administrators, who might not be versed in developing programs, will learn the rationale for setting up profiles in certain ways, and that application programmers writing the servers will be able to document the security requirements of their products.

Appropriate decisions need to be made regarding server security. In the past, applications had to run as APF-authorized to be able to call RACF to build task-level security. z/OS UNIX provides services for servers written in C to create task-level security without being APF-authorized. This chapter describes how a server can create thread-level security environment and how to control which servers have the ability to do so. It also describes the procedures for preparing a z/OS system for a server that uses thread-level security for its clients. (Note that a thread on UNIX systems corresponds to a task on MVS; so, thread-level security is the same as task-level security.)

The previous chapter, Chapter 17, is a prerequisite to this chapter and discusses topics that also relate to setting up thread-level security. If you need a high level of security, read and follow the steps in that chapter and this chapter

Designing security for servers

This section is intended for the application developer who is designing and developing servers that use z/OS UNIX. The section describes:

- Setting up the clients with the appropriate security; see “Setting up threads and security.”
- Controlling access to resources; see “Checking authority to use protected resources” on page 321.
- Using the RACF client ACEE support; see “Limitations of RACF client ACEE support” on page 322.
- Writing the documentation that supports your server; see “Documenting the security requirements” on page 322.

Setting up threads and security

z/OS UNIX supports two fundamental types of application servers: multithreaded servers and single-threaded servers.

- A multithreaded server has multiple sequential flows of control. In this family of applications, the server can process more than one unit of work at a time.
- A single-threaded server has one sequential flow of control. In this family of applications, the server processes one unit of work at a time.

z/OS UNIX provides the `pthread_security_np()` callable service and support through the C run-time library. It enables unauthorized multithreaded servers to create and delete a RACF security environment in a way that is mediated and controlled by the kernel and RACF. Multithreaded servers can customize the security environment of a thread, thus allowing it to be executed under a different RACF identity than that of the server. You must authorize the server to use that service.

The term *unauthorized* refers to applications that are not APF-authorized and do not run in supervisor state or in a system storage protection key.

A server that uses the `pthread_security_np()` service can customize the RACF identity of a thread. For example, a DCE server on z/OS could accept requests through DCE remote procedure call (RPC). This server initiates a thread that processes the client's request. If the server customizes the thread initiated for the client with the client's RACF identity, any resource access decisions to RACF-protected resources are made using the client's RACF identity and authorizations.

Depending on the trust you place in a server, you have the option of enforcing whether to use both the server's RACF identity and the RACF identity of the client in resource access control decisions on z/OS.

You can choose one of the following:

- Only the RACF user ID of the client is used in local resource access control decisions made by RACF on z/OS.
- Both the RACF user ID of the server and the RACF user ID of the client are used in local resource access control decisions on z/OS.

The use of the `pthread_security_np()` service is in part protected by the RACF FACILITY class profile BPX.SERVER. If this profile is defined, then the RACF user ID that is associated with the server needs at least READ authority to use the `pthread_security_np()` service.

- If the RACF user ID that is associated with a server is permitted with UPDATE access to this profile, the server is allowed to establish a thread-level (task-level) security environment for clients connecting to the server. With UPDATE authority to BPX.SERVER in the RACF FACILITY class, the server can act as a *surrogate* of the client. This means that the identity of the thread associated with the request from the server's client executes with the z/OS user ID of the server's client.

The RACF identity of the client determines the type of access allowed to z/OS resources (such as data sets) and z/OS UNIX resources (such as HFS files), which are accessed by the client's thread in the server.

- READ access allows the server to establish a thread-level security environment for the clients it services. However, the user ID of the server and the user ID of the client must be authorized to the resources the server accesses. A thread-level security environment in which both the client's and server's identities are used in the access control decision, but a password was not supplied by the client, is called an *unauthenticated client* security environment.

Depending on the design and implementation of the client/server application, a client might need to supply an authenticator to the server.

For example, the client might be prompted to supply a password or a password substitute, such as a RACF PassTicket, to the server to prove its identity. If a RACF password or PassTicket is specified as an option on the `pthread_security_np()` service, and the password or PassTicket is valid for the client user ID, only the RACF user ID of the client is used in rendering access control decisions. This task level security environment created by a server is called an *authenticated client* security environment. Because the client has trusted the server sufficiently to supply a RACF password or PassTicket to the server, the server can act as a surrogate for that client.

This capability enables you to determine on behalf of which user IDs the server can act and what resources the server can access when acting on behalf of one of its clients.

Potentially, for additional security checking, two audit records can be produced to audit the client accessing the resource and the server accessing the resource on behalf of the client.

If you choose to implement this additional security checking, you might need to authorize the identity associated with the server to the resource profiles that protect the resources accessed by the server on behalf of its clients.

Application services and security for DCE clients

Through z/OS UNIX, the C run-time library, and RACF, two services are available that enable servers on z/OS to map a DCE identity to a RACF user ID or map a RACF user ID to a DCE identity and to invoke RACF authorization services.

The `convert_id_np` (BPX1CID) callable service is the z/OS UNIX service. It converts a DCE principal's UUID pair (cell UUID and principal UUID) to the RACF user ID that has been cross-linked with the UUID pair. This service also accepts a RACF user ID and returns the corresponding DCE UUIDs that have been cross-linked with the RACF user ID. This z/OS UNIX service is also supported through the C run-time library via the `__convert_id_np()` function call.

These services use the SAF callable service, `R_dceruid` (IRRSUD00), which accesses the appropriate profile information stored in the RACF database, to perform the identity conversion. The use of these identity mapping functions is RACF-protected. The `R_dceruid()` callable service performs an authorization request to determine if the user ID associated with the server is authorized to use the identity conversion service. Controlling the use of these conversion services is discussed in "Documenting the security requirements" on page 322.

For more information about the `convert_id_np` (BPX1CID) callable service, see *z/OS UNIX System Services Programming: Assembler Callable Services Reference*. The C language support for `__convert_id_np()` is discussed in *z/OS C/C++ Run-Time Library Reference*.

Checking authority to use protected resources

Application developers might want a server to check the authority of a user to access RACF-protected resources. In this way, the server could control access to those resources. The resources include printers and tapes, but not HFS files and directories and MVS data sets. Use the z/OS UNIX `auth_check_resource_np` (BPX1ACK) callable service or the C run-time library `check_resource_auth_np()` function call to invoke the RACF `v_dceauth` callable services to do the necessary

checking. The resources must be defined to RACF general resource classes. The server must have read access to the BPX.SERVER FACILITY class profile or have UID(0); in addition, all server modules must be defined to RACF.

Limitations of RACF client ACEE support

If both the server's RACF identity and the client's RACF identity are used to make access decisions, you should be aware of limitations of the RACF client ACEE support.

- RACROUTE REQUEST=FASTAUTH processing does not check both the server and client RACF identities automatically.

Unauthorized servers cannot use the RACROUTE REQUEST=LIST instruction to build in-storage profiles for RACF defined resources. Profiles must reside in storage before RACROUTE REQUEST=FASTAUTH can verify a user's access to a resource.

- The client/server relationship is not propagated from the server.

If your server controls access to resources by checking and authenticating both the server's RACF identity and client's RACF identity, treat servers you do not trust as *end points* on z/OS. These servers should not be allowed to submit batch jobs or use the services of other servers that run exclusively under the identity of the client. You must ensure that servers that do not meet this criteria are not authorized to the profile BPX.SERVER in the RACF FACILITY class.

Documenting the security requirements

In documentation that accompanies your servers, you might need to give some instructions to the security administrator whose installation will be running the server. This might happen if your server uses services that require special authority; these services include:

- The SAF R_dceruid() callable service
- The z/OS UNIX convert_id_np callable service
- The C library function __convert_id_np() function call

Without the appropriate authority set up at the installation, your server will not run. Documentation that accompanies these services tells the security administrator the kind of RACF definitions to set. For example, if the server uses the z/OS UNIX convert_id_np() callable service, the server must have READ access or higher to the IRR.RDCERUID FACILITY class profile.

Establishing the correct level of security for servers

The choice of security level is a decision more likely made by management than by security administrators. That decision depends on answers to the questions "How secure does our company's information need to be?" and "How much do we trust our employees?" Regardless of who makes the decision, it is important that both application developers and security administrators understand the two levels of security supported by z/OS, and the differences between them. The two levels are: UNIX level and z/OS UNIX level. Read the following descriptions to help you decide which level of security is appropriate for your server.

UNIX level: BPX.SERVER is not defined

If the BPX.SERVER (or BPX.DAEMON) FACILITY class is not defined, your system has UNIX-level security. In this case, the system is less secure. This level of security is for installations where superuser authority has been granted to system programmers. These individuals already have permission to access critical MVS

data sets such as PARMLIB, PROCLIB, and LINKLIB. These system programmers have total authority over a system. Server programs that run with superuser authority can issue `pthread_security_np()` service to change the MVS identity of a thread.

To establish UNIX-level security, assign a UID of 0 to the superuser and assign a UID of 0 to the user ID used for running server programs; for example, DATASVR.

z/OS UNIX level: BPX.SERVER is defined

There are two z/OS UNIX levels:

- RACF running with enhanced program security, BPX.SERVER defined, and BPX.MAINCHECK defined. You can use BPX.MAINCHECK for any privileged z/OS UNIX application that requires a program controlled environment, because the application uses a privileged z/OS UNIX service that requires one. An example is the `__passwd()` service, which is used by applications such as telnet and rlogin.
- BPX.SERVER

RACF with enhanced program security, BPX.SERVER, and BPX.MAINCHECK

If you enable enhanced program security, and you have any daemons or servers that run execute-controlled programs (MVS programs defined to RACF in the PROGRAM class using EXECUTE authority, or loaded from libraries using EXECUTE authority), then you must define the initial program executed by your daemon or server as a trusted ("MAIN") program to RACF via the PROGRAM class. If this initial program resides in the z/OS UNIX file system, rather than in an MVS library, you will need to move it to an MVS library.

Additionally, you can choose whether to extend the enhanced program security protection to your UNIX daemons and servers that do not make use of RACF execute-controlled programs. You would enable this function by defining the profile BPX.MAINCHECK to RACF in the FACILITY class. Again, you would need to ensure that the initial program executed by your daemon or server resides in an MVS library and you would need to define it to RACF as a PROGRAM with the MAIN attribute.

Kernel services that change a caller's z/OS user identity require the target z/OS user identity to have an OMVS segment defined. If you want to maintain this extra level of control at your installation, you will have to choose which daemons to permit to the BPX.DAEMON FACILITY class. You will also have to choose the users to whom you give the OMVS security profile segments. To accomplish this, refer to "Steps for preparing the security program for daemons" on page 294.

"Steps for setting up enhanced program security" on page 300 explains how to set up enhanced program security.

BPX.SERVER

If BPX.SERVER (or BPX.DAEMON) FACILITY class is defined, your system has z/OS UNIX-level security. In this case, the system is more secure than a traditional UNIX system.

This level of security is for customers with very strict security requirements who need superusers to maintain the file system but who do not want these users to

have the authority to change their identities to access existing MVS resources. To accomplish this, take the additional steps described in “Defining servers to use thread-level security.”

Defining servers to use thread-level security

When the profile BPX.SERVER is defined, there may be two authorization checks:

- The first check authorizes the use of the `pthread_security_np()` service.
- The second check authorizes for whom the server can establish a security context. This check establishes the scope of users for whom the server can act as a *surrogate*. See “Defining servers to process users without passwords” on page 326 for the steps required to enable servers to act as surrogates for their clients when a password is not specified on the `pthread_security_np()` service.

You can also use the BPX.SERVER profile to set the scope of z/OS resources that the server can access when acting as a surrogate for its clients. There are two levels of authority that can be granted to the server using thread-level security services:

- **UPDATE access**

Lets the server establish a thread-level (task-level) security environment for clients connecting to the server. When the RACF identity of the server has been granted UPDATE authority to BPX.SERVER in the RACF FACILITY class, the server is capable of acting as a *surrogate* for the client. This means that the identity of the thread associated with the request from the server's client runs with the z/OS user ID of the server's client. Access control decisions to z/OS resources (such as data sets) and to z/OS UNIX resources (such as HFS files) which are accessed by the client's thread in the server are made using the RACF identity of the client.

- **READ access**

Lets the server establish a thread-level security environment for the clients that it services. However, the user ID of the server and the user ID of the client must be authorized to the resources which the server will be accessing. A thread-level security context in which both the client's and server's identity is used in the access control decision and a password was not supplied by the client is called an *unauthenticated client* security context.

Depending on the design and implementation of the client/server application, a client may have to supply an authenticator to the server. For example, the client may be prompted to supply a password or a password substitute, such as a RACF PassTicket to the server to prove its identity. If a RACF password or PassTicket is specified as a parameter on the `pthread_security_np()` service, and the password or PassTicket is valid for the client user ID, even if the server's identity has been granted READ access to the profile BPX.SERVER in the RACF FACILITY class, the task level security environment is only used in access control decisions. That is, only the RACF user ID of the client is used in making access control decisions. This task level security environment created by a server is called an *authenticated client* security context. Since the client has trusted the server sufficiently to supply a RACF password (or PassTicket) to the server, the server is granted the capability of acting as a surrogate for that client (user).

Server setup

This section shows how to set up servers. The following steps are for a sample server called DATASVR. As you add more servers, you will need to follow similar procedures.

1. All programs that are loaded into an address space requiring server authority (including the server program and any run-time library modules) need to be marked as controlled.

To identify programs as controlled, use the RACF RDEFINE command. For a discussion on identifying controlling programs, see “Customizing the system for IBM-supplied daemons” on page 295.

2. The servers need to be assigned a user ID.

Assume that the user ID of the server is DATASVR. Define user ID DATASVR to RACF:

```
ADDUSER DATASVR DFLTGRP(OMVSGRP) OMVS(UID(7) HOME('/'))  
PROGRAM('/bin/sh')) NOPASSWORD
```

You can use the NOPASSWORD option with the ADDUSER command for DATASVR. This indicates that it is a protected userid that cannot be used to enter the system by means of a password. The userid will not be revoked due to invalid logon attempts.

In this case, you are defining the DATASVR user ID without a TSO/E segment.

3. Create a cataloged procedure:

```
//DATASVR PROC  
//DATASVR EXEC PGM=DATASVR,REGION=0M,TIME=NOLIMIT,  
//      PARM='POSIX(ON) ALL31(ON)/ serverparms'  
//SYSPRINT DD SYSOUT=*
```

4. In order for this DATASVR cataloged procedure to obtain control with the desired user identity, you must either add it to the RACF STARTED class or add an entry to the started procedures table.

For information on using the STARTED class to define started procedures, see *z/OS Security Server RACF Migration*.

To add an entry to the started procedures table:

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

5. To create the server FACILITY class profile, issue:

```
RDEFINE FACILITY BPX.SERVER UACC(NONE)  
SETROPTS RACLIST(FACILITY) REFRESH
```

6. If you have not already done so for daemon support, activate Program Control for servers by issuing:

```
SETROPTS WHEN(PROGRAM)
```

7. The next decision that must be made is the level of authority to be granted to the server using thread-level security services. The BPX.SERVER FACILITY class profile controls the server's access to the pthread_security_np() service. There are two choices when setting the server's authority:

- a. UPDATE access allows the server to establish a thread-level (task-level) security environment for clients connecting to the server. Decisions about access control for z/OS resources (such as data sets) and to z/OS UNIX resources (such as HFS files) that are accessed by the client's thread in the server are made using only the RACF identity of the client.

To give UPDATE access in the BPX.SERVER FACILITY class profile to user ID DATASVR:

```
PERMIT BPX.SERVER CLASS(FACILITY) ID(DATASVR) ACCESS(UPDATE)  
SETROPTS RACLIST(FACILITY) REFRESH
```

- b. READ access allows the server to establish a thread-level security environment for the clients that it services. However, unless the server has specified a valid RACF password or PassTicket on the `pthread_security_np()` service invocation, the user ID of the server and the user ID of the client are used in resource access control decisions. Following is the PERMIT command to give DATASVR server authority for unauthenticated clients:

```
PERMIT BPX.SERVER CLASS(FACILITY) ID(DATASVR) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

8. If you are installing a product that uses thread-level security services, check the documentation that is supplied with the product to determine if the server requires READ or UPDATE access to the BPX.SERVER profile.

If you grant READ access to the BPX.SERVER profile in the FACILITY class, and the server does not request a password or PassTicket for its clients, both the server's user ID and the client's user ID are used in decisions about resource access control. Additional security administration will have to be performed to ensure that both the server's user ID and the client's user ID were appropriately authorized to the resources that are accessed by the server.

For the unauthenticated client case, see 7b.

9. To start DATASVR, issue the following command from the MVS console:

```
S DATASVR
```

If the DATASVR daemon is deactivated, you can also issue this command to restart it.

Defining servers to process users without passwords

Depending on the design and implementation of a client/server application, a client may not supply an authenticator to the server. For example, some servers process user requests that come from generic user IDs representing anonymous users, or use a method of authentication other than a user ID and password combination.

In this case, in which the RACF password or password substitute (such as the RACF PassTicket) is not specified on the `pthread_security_np()` service invocation, an additional check is made to ensure that the server is authorized to act as the client. z/OS UNIX uses profiles defined to the RACF SURROGAT class to authorize the server to act as a surrogate of a client. Profiles defined to the SURROGAT class are of the form:

```
BPX.SRV.<userid>
```

<userid> is the MVS user ID of the user that the server will act as a surrogate of. See "Defining servers to use thread-level security" on page 324 for the steps to authorize a server to act as a surrogate for client user IDs.

Some servers have the requirement to process user requests that come from generic user IDs representing anonymous users. In order for servers to process requests for thread-level security without passwords, follow the steps shown below.

The following steps are for a sample server called DATASVR that can support user ID ANONYMOS without a password. As you add more servers, you will need to follow similar procedures.

1. Activate the SURROGAT class support in RACF:

```
SETROPTS CLASSACT(SURROGAT)
```


The activation has to be done only once on the system. The SURROGAT class may already have been set up on your system.

If a daemon or server you are running will use the SURROGAT support, consider using the RACLIST command to keep the SURROGAT profiles in storage. The following example shows how to cache the SURROGAT profiles in storage:

```
SETOPTS RACLIST(SURROGAT)
```

2. If the SURROGAT profile is in the RACLIST, any changes to the SURROGAT profiles must be followed by a REFRESH command. To create the SURROGAT class profile for user ANONYMOS, issue:

```
RDEFINE SURROGAT BPX.SRV.ANONYMOS UACC(NONE)  
SETOPTS RACLIST(SURROGAT) REFRESH
```

A similar SURROGAT profile is required for each user ID that a server must support without a password.

3. To permit server DATASVR to create a thread-level security environment for user ANONYMOS, issue the PERMIT command:

```
PERMIT BPX.SRV.ANONYMOS CLASS(SURROGAT) ID(DATASVR) ACCESS(READ)  
SETOPTS RACLIST(SURROGAT) REFRESH
```

Chapter 19. Monitoring the z/OS UNIX environment

NOT Programming Interface information

Overview

Your z/OS system can monitor performance, use of resources, and the use of system resources by users and programs.

Use the information that is collected to tune the system. This can improve performance and reduce resource consumption.

Reporting on activities using SMF records

See *z/OS MVS System Management Facilities (SMF)* for the full contents of SMF records provided for z/OS UNIX and for information on how to obtain the records.

You can use SMF to report on activity from a user application, to report activity on a job and jobstep basis, and to report activity of mounted file systems and files.

Using 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

This process section also provides information on file system lookups, which can use significant resources on systems with hierarchical files.

You can monitor the file system activity of various classes of users by postprocessing SMF type-30 records. This may be helpful in forecasting DASD and other system resource requirements. If key jobs appear to be doing many lookups, your installation may be able to reduce this overhead. To reduce the overhead, reorganize the file system so that key files are closer to the root of the file system.

Applications can also reduce lookup activity by using the **chdir** command to change the working directory and specifying only the filename when opening a file.

SMF records also contain a program name field for job steps that are initiated by `fork()`, `spawn()`, or `exec()`. For interactive commands, this allows performance analysts to determine what resources were required to complete a particular command.

If a user runs the OMVS command with the SHAREAS option or sets the environment variable `_BPX_SHAREAS` to YES, two or more processes can be running in the same address space. In this case, SMF provides process identification only for the first process in the address space. However, resource consumption is accumulated for all processes that are running.

With an exec that follows a `setuid()`, the exec processing no longer creates a substep. Instead, the initiator stops the old job (ending type 30 record). Then a new job is started with the user ID that was established on the `setuid()`.

Preventing SMF Record Type 34 and 35 for SUBSYS OMVS

When a new address space is created as a result of a `fork()` or `spawn()` service, SMF cuts a Type 34 record. When the process ends, SMF cuts a Type 35 record. Type 34 is defined as TSO Logon and Type 35 is defined as TSO logoff. If you do not have Type 34 or Type 35 active, you do not need to take any further action. If you do use Type 34 and Type 35 for TSO accounting, then you need to suppress these recordings for UNIX processes. To suppress these records, add the following:

```
SYS(TYPE(34,35))  
SUBSYS(OMVS,NOTYPE(34,35))
```

Using SMF Record Type 74

SMF record type 74, subtype 3, reports kernel activity. For more information, see *z/OS MVS System Management Facilities (SMF)*.

Using SMF Record Type 80

SMF record type 80 includes an extended-length relocate section. For specifics on auditing information in SMF record type 80, see *z/OS Security Server RACF Macros and Interfaces*.

Using SMF Record Type 92

SMF record type 92 reports 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.

Unmount records also provide the following I/O data summarized for the entire mountable file system:

- Directory reads
- Read and write callable services requested
- Read and write EXCP counts
- Total bytes read and bytes written

File-close records provide information on I/O activity of a user or application against a specific file. These records provide the following data for a specific user or application and a specific file:

- Read and write callable services requested
- Read and write EXCP counts
- Total bytes that are read and bytes that are written
- Pathname of the file

Collecting SMF type 92 open and close records can become expensive. These subtypes (subtypes 10 and 11) should only be collected when file-level data is needed.

To collect information about the activity of an HFS mounted file system, you must be collecting SMF type 92 records with subtype 5 at the time the file system is mounted and unmounted.

To collect information on the activity of a specific HFS file, you must be collecting SMF type 92 records with subtype 11 at the time the file is opened and closed.

When an HFS file system is mounted, SMF begins collecting accounting data for the file system. When an HFS file is opened, SMF begins collecting accounting data for the file. Partial SMF accounting does not occur; either all the information for a file system or an open file is collected, or none is collected.

If you turn off SMF recording temporarily, data collection is still done. However, if you turn off SMF recording and then unmount a file system, the data is lost. Similarly, if you turn off SMF recording and then close a file, the data is lost. To obtain a record of activity, SMF recording must be active at the time a file system is both mounted and unmounted. Or it must be active at the time a file is both opened and closed.

Monitoring process activity

Using installation exits

You can use installation exits to enable applications to monitor z/OS UNIX process activities.

Pre-process initiation exit (BPX_PREPROC_INIT)

Receives control immediately before the creation of any new z/OS UNIX process. This exit cannot use any z/OS UNIX callable service. When these exit routines receive control, the Process Exit Data Block (PEDB) will contain data about the creating process.

Post-process initiation exit (BPX_POSPROC_INIT)

Receives control immediately after the creation of any new z/OS UNIX process. When this exit receives control, the Process Exit Data Block (PEDB) will contain the creator and the new process data.

Process image initiation exit (BPX_IMAGE_INIT)

Receives control immediately before any new z/OS UNIX process image is initiated. This occurs after a successful spawn, attach_exec, attach_execmvs, exec, or execmvs callable service is done. The exit will receive control before the new process image file is run. When this exit receives control, the Process Exit Data Block (PEDB) will contain the data of the creator and the new image.

Pre-process termination exit (BPX_PREPROC_TERM)

Receives control immediately before the termination of a z/OS UNIX process. These exits may receive control in the address space of the process or in the master address space, if the address space of the process was terminated. In the latter case (ASID=1), the exit cannot use z/OS UNIX callable services. When these exits receive control, the Process Exit Data Block (PEDB) will contain details about the terminating process.

Exit routines can be added to each exit point. z/OS UNIX passes control to the exit routine when an exit point is reached. Information about the current process and its creator is then passed to the exit routine.

Defining exits

The z/OS UNIX kernel defines the four process start/end exits at kernel initialization time via the CSVDYNEX service.

Rule: When you are adding exit routines to an exit, certain exit attributes are required.

For BPX_PREPROC_INIT, BPX_POSPROC_INIT, and BPX_IMAGE_INT:

```
AMODE=31
REENTRANT=REQ
PERSIST=IPL
ABENDNUM=10000
ABENDSCONSEC=YES
FASTPATH=YES,KEY=0
```

For BPX_PREPROC_TERM:

```
AMODE=31
REENTRANT=REQ
PERSIST=IPL
ABENDNUM=10000
ABENDSCONSEC=YES
FASTPATH=NO,KEY=0
```

Adding exit routines to exits

You can use any of the following methods to add exit routines to exits:

- PROGxx parmlib member
- SETPROG console command
- REQUEST=ADD via the CSVDYNEX service

Example: To add the DUBEXIT exit routine to the BPX_PREPROC_INIT exit via a PROGxx member:

```
EXIT ADD
  EXITNAME(BPX_PREPROC_INIT)
  MODNAME(DUBEXIT)
```

Example: To remove the DUBEXIT exit routine from the BPX_PREPROC_INIT exit:
SETPROG EXIT,DELETE,EXITNAME=BPX_PREPROC_INIT,MODNAME=DUBEXIT,FORCE=YES

_____ **End of NOT Programming Interface information** _____

Chapter 20. Tuning performance

Overview

You need to take some tuning steps because you are combining MVS and UNIX. There are two tuning situations, depending on how your system is being used: as a production system or a porting system. For both, you can take important steps to improve performance and control resource consumption.

To learn how to improve performance on a porting system, read Chapter 8 of *Porting Applications to the z/OS UNIX Platform*.

Adjusting storage size

If your system is running in a virtual server or as a VM guest, the storage size should be at least 64M.

Using DASD cache

Place on cached DASD:

- Volumes that contain user file systems. To get the responsiveness that UNIX users are accustomed to, place these volumes on cached DASD that has SET DASD FAST WRITE on. Do this because the hierarchical file system hardens all file system data to disk synchronously. (Data is stored on disk synchronously on certain writes, and any remaining data is stored to disk synchronously on close. That is, the close does not return to the user until the file has been completely stored on disk.)
Tip: To avoid DASD contention, the user file systems should be distributed among multiple control units and DASDs. Too many user file systems on one volume can negatively affect I/O performance. See “Improving performance of file systems” on page 343 for more information on user file systems.
- The RACF data base.

Improving performance of run-time routines

When C programs (including the shell and utilities) are run, they frequently use routines from the Language Environment run-time library, which come from the SCEERUN data set. On average, about 4 MB of the run-time library are loaded into memory for every address space running a Language Environment-enabled program, and copied on every fork. If you have 200 address spaces running, this uses 800 MB of pageable storage. It also increases your paging rates or reduces the amount of work that the system can support. For information about the effect of putting modules into the LPA, see *z/OS MVS Initialization and Tuning Guide*.

The following sections describe how you can reduce this overhead and improve performance.

Placing SCEERUN in the link pack area

Because the SCEERUN data set has many modules that are not reentrant, you cannot place the entire data set in the link pack area (LPALSTxx parmlib). The SCEELPA data set contains a subset of the SCEERUN modules—those that are re-entrant, reside above the line, and are heavily used by UNIX System Services. (For more information, see *z/OS Language Environment Customization*.)

If you put the SCEERUN data set in the link list (LNKLSTxx), you can place the new SCEELPA data set in LPA list. Doing this will improve performance.

You can also add additional modules to the LPA, using the Modify Link Pack Area (MLPA=) option at IPL. You can also use the Dynamic LPA capability (SET PROG=). Using the Dynamic LPA method avoids the performance degradation that occurs with the use of MLPA.

The RUNOPTS parameter in the BPXPRMxx parmlib member specifies the Language Environment run-time options that is to be passed to */etc/init* when using RTLS.

Placing SCEERUN in the link list

If you choose not to put any modules from SCEERUN in the LPA, you can still put SCEERUN in the link list. This will not perform as well as having modules in LPA, but can still benefit from reduced input/output due to management by LLA and VLF.

Managing the run-time library with RTLS

Some installations cannot put the current level of the Language Environment run-time library into the LINKLIST because older Language Environment 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 enables installations to use more than one level of the run-time library on the same system without using STEPLIBs. They can put key run-time library modules from more than one level of Language Environment into common storage for shared access.

See *z/OS Language Environment Customization* for information on using RTLS on your system. You will need to set up some FACILITY profiles, as documented in the CSVRTLxx description in *z/OS MVS Initialization and Tuning Reference*.

After you set up RTLS, you only need to set up RUNOPTS in the BPXPRMxx member for most z/OS UNIX environments. (Customizing the BPXPRMxx member is discussed in “Customizing the BPXPRMxx parmlib members” on page 24.

For z/OS UNIX users to use RTLS, you must also specify RTLS(ON), LIBRARY, and, optionally, VERSION run-time options in the RUNOPTS parameter of the BPXPRMxx parmlib member. For example:

```
RUNOPTS(RTLS(ON) LIBRARY(xxxxxxxx) VERSION(yyyyyyyy))
```

where xxxxxxxx is the library name and yyyyyyyy is the version name assigned in the CSVRTLxx parmlib member for the current level of Language Environment (for example, CEE.SCEERUN).

Managing the run-time library in STEPLIBs

If you decide not to put the run-time library in the link list or RTLS, then you must set up the appropriate STEPLIB for each application that needs to load modules from SCEERUN. Although this method always uses additional virtual storage, you can improve performance by defining the SCEERUN data set to LLA. This reduces the I/O that is needed to load the run-time modules.

Improving compiler performance

This section discusses how you can improve compiler performance by placing the C/C++ compiler and Program Management Binder in the LPA.

Putting compiler load modules into LPA

On systems where application development is the primary activity, you might be able to improve performance if you put CBC.SCCNCMP or CBC.SCBCCMP (or both) in the LPALST concatenation. All compiler modules run above the line and they consume just over 42 MB in total.

Place the program binder in LPA:

- From SYS1.LINKLIB:

Module	Location
IEFIB600 (alias IEFXB603)	44K below the line
IEWBLINK	2K below the line

IEWBLINK has these aliases:

- alias HEWL
- alias HEWLDRGO
- alias HEWLH096
- alias HEWLOAD
- alias HEWLOADR
- alias IEWBLDGO
- alias IEWBLOAD
- alias IEWBLODI
- alias IEWBODEF
- alias IEWL
- alias IEWLDRGO
- alias IEWLOAD
- alias IEWLOADI
- alias EWLOADR
- alias LINKEDIT
- alias LOADER

- From CEE.SCEERUN:

Module	Location
EDCRNLIB (alias EDCRNLIST)	Above the line

Caching RACF user and group information in VLF

Caching UIDs and GIDs improves performance for commands such as **Is -I**, which must convert UID numbers to user IDs and GID numbers to RACF group names. RACF allows you to cache UID and GID information in Virtual Lookaside Facility (VLF). 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(IRRSMP)
  EMAJ(SMAP)
```

For details about these VLF and the other VLF classes that are used by RACF, see *z/OS Security Server RACF System Programmer's Guide*.

Start VLF, specifying the updated member (in this example, COFVLF33 member) with an operator command:

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

Because VLF is started after RACF and OMVS, you may get a message from RACF during the IPL saying that running without VLF will cause slower performance. If VLF is being started, you can ignore this message.

For information about updating the VLF parmlib member COFVLFxx, see "COFVLFxx parmlib member to activate RACF classes" on page 39.

Moving HFS executables into the link pack area

Some executables in the HFS may be commonly used by many concurrent users, or they may be loaded and deleted frequently during normal production. Such executables are performance sensitive, and they may be good candidates for inclusion in the LPA. Moving such programs to the LPA can reduce storage consumption, reduce DASD I/O activity for loads, and reduce the storage copied on each fork().

One thing to consider when you analyze which HFS executables belong in LPA is that modules with the sticky bit on are not eligible for local spawn(). If your executable is normally invoked by spawn(), either by the shell or by another application, turning on the sticky bit forces spawn() processing to execute the program in a spawned child address space. In cases where local spawn() would be used if the sticky bit were not on, this reduces the benefit of loading the executable from the LPA.

To move an executable in the HFS into the LPA, do the following steps:

1. If the executable or DLL name is less than 8 characters excluding the extension (such as longname.dll):
 - a. Bind the executable or DLL into a PDS (for example, LONGNAME)
 - b. For the executable or DLL in the HFS, turn on the sticky bit. For example:

```
chmod +t longname.dll
```
 - c. If the executable or DLL name has invalid characters, then do a symbolic link such as:

```
ln -s longname long+name
```
2. If the executable or DLL name is more than 8 characters long, excluding the extension (for example, reallylonglongname.dll):
 - a. Bind the executable or DLL into a PDS (for example, REALLY)
 - b. Create an external link for the name. For example:

```
ln -e REALLY reallylonglongname.dll
```

To bind the executable or DLL into a PDS, you can use the following sample JCL:

```

//PUTINLPA JOB MSGLEVEL=(1,1)
//*
/* INLMOD DD STATEMENT SPECIFIES THE DIRECTORY THAT CONTAINS *
/* THE PROGRAM. *
/*
/* THE INCLUDE STATEMENT SPECIFIES THE NAME OF THE FILE TO *
/* RUN FROM THE LPA. *
/*
/* THE NAME STATEMENT SPECIFIES THE FILE NAME BUT IN *
/* UPPERCASE. THIS MUST BE SAME AS THE FILE NAME. *
/*
/*
//LINK EXEC PGM=IEWL,REGION=100M,
// PARM='LIST,XREF,LET,RENT,REUS,AMODE=31,RMODE=ANY,CASE=MIXED'
//SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(10,10))
//SYSPRINT DD SYSOUT=*
//INLMOD DD PATH='/bin/'
//SYSLMOD DD DSN=OECDMD.LPALIB,DISP=SHR
//SYSLIN DD *
        INCLUDE INLMOD(myprog)
        ENTRY CEESTART
        NAME MYPROG(R)
/*

```

Figure 54. Job for placing a program in the LPA

You should use an SMP/E usermod to link any IBM-supplied programs from an HFS into another library. (For example, in order to load it into LPA.) Doing so automatically keeps the two copies of the module at the same level when service is installed. It also provides a record of modifications to your systems. See *SMP/E User's Guide* for more information about SMP/E usermods.

Also, not all modules are eligible for LPA. Modules placed in LPA must be both reentrant and executable. For more information, see *z/OS MVS Initialization and Tuning Reference*.

Using the shared library extended attribute

Shared object libraries contain subroutines that can be shared by multiple processes. Programs using shared libraries contain references to the library routines that are resolved by the loader at run time. Shared library modules are in the library region in shared virtual storage. The shared library region is carved out of each address space that do dillloads against HFS programs (DLLs) that have the `st_sharelib` extended attribute turned on. This region is then shared between all the address spaces that load these DLLs.

Executables that have the `st_sharelib` extended attribute turned on are called *system-shared library programs*. They are an optimal way of sharing large HFS executables across many address spaces in the system. These executables are shared on a megabyte boundary to allow for the sharing of a single-page table (similar to LPA). The storage used in the user address space to establish the mapping to the shared library region is from the high end of private storage; it does not interfere with the virtual storage used by the application program.

Executables that have the `.so` suffix in their file names are called *user-shared library programs*. They are optimal for sharing HFS executables across a smaller set of similar user address spaces in the system. These executables are shared on a page boundary. The storage used in the user address space to establish the mapping to the shared library region is from the low end of private storage; it comes out of the same user region storage used for private area loaded modules.

See “Defining HFS files as shared library programs” on page 299 for information about setting the `st_sharelib` extended attribute.

Tuning limits in parmlib

This section contains information that may be helpful in tuning your z/OS UNIX environment. It provides guidelines that should prove to be generally helpful. However, because each installation is unique, some of the recommendations may not be appropriate for your system.

For more information, refer to these documents:

- *z/OS MVS Planning: Workload Management*
- *z/OS MVS Initialization and Tuning Guide*
- *z/OS MVS Initialization and Tuning Reference* for parmlib members
- *z/OS RMF User's Guide* for RMF monitoring
- *z/OS RMF Report Analysis* for RMF reports

Monitoring BPXPRMxx parameter limits

You can monitor the status of the z/OS UNIX system and process limits with the `D OMVS`, `LIMITS` operator command and console messages that indicate when limits are reaching critical levels.

You can then use `SET OMVS` or `SETOMVS` to change certain system limits dynamically, or `SETOMVS` with `PID=` to change a process-level limit for a specific process. See *z/OS MVS System Commands*.

The `LIMMSG` statement in the `BPXPRMxx` parmlib member controls message activity for limits checking. You can specify whether no console messages are to be displayed when any of the parmlib limits have been reached (`NONE`); console messages are to be displayed for all processes that reach system limits and for certain process limits (`SYSTEM`); or console messages are to be displayed for all the system limits and the process limits (`ALL`).

The `LIMMSG` options can be changed with the `SETOMVS LIMMSG` command. The `LIMMSG` value appears in the `D OMVS,O` display.

If the `LIMMSG` statement is specified with `SYSTEM` or `ALL`, a warning console message appears whenever a limit reaches 85%, 90%, 95%, and 100%, identifying the process that has reached the limit. When the limit reaches the next limit level, the prior message is removed from the console and a new message indicates the new limit level that has been reached. When the limit falls below the 85% threshold, a message indicates that the resource shortage has been relieved.

Changing from `LIMMSG(ALL)` or `LIMMSG(SYSTEM)` to `LIMMSG(NONE)` with the `SETOMVS` command stops any further monitoring of resources. However, existing outstanding messages are not deleted from the screen for a process until the limit is relieved for that process.

Tip: When `LIMMSG(ALL)` is in effect, a large number of messages can be issued. This option is best suited for use during the initial configuration of a system, when the installation has not yet determined the optimal settings for the z/OS UNIX parmlib limits.

Tuning process activity

z/OS UNIX provides the system programmer with a number of controls that monitor and tune the use of system resources by users. This section focuses on the following fields in the BPXPRMxx statements:

- MAXUIDS
- MAXPTYs
- MAXPROCUSER
- MAXPROCSYS

Initial rules of thumb

1. Assume that each user will consume up to double the system resources required for a TSO/E user.
2. Assume that at most 4 PTYs will be required per average user.
3. Assume that the starting point for maximum processes per user is 25.
4. Assume that 4 concurrent processes will be required by the average active user.
5. Assume that 5 processes will be required for various daemons.
6. Assume that 3 concurrent address spaces will be required by the average active user. This number will be high if your users are running with the `_BPX_SHAREAS` environment variable set to YES.

If you have a few users who need a large number of processes, you should set the process limits for these users by using the PROCUSERMAX keyword in the OMVS segment.

Example

Assume that your system supports 600 TSO/E users and has enough capacity for 20 additional users. Rather than adding more TSO/E work, you want to allow TSO/E users to access z/OS UNIX. You have no other z/OS UNIX work on your system at this time.

In this example, in BPXPRMxx, the initial settings might be:

```
MAXUIDS(20)
MAXPTYs(80)
MAXPROCUSER(25)
MAXPROCSYS(85)
```

MAXUIDS 20 - If you allow 20 current TSO/E users to access the z/OS UNIX system, each of them could consume twice the resource they normally used for TSO/E. This would require all your remaining system resources.

MAXPTYs 80 - Assume that 4 PTYs are needed per user. Users can login with multiple sessions at the same time.

MAXPROCUSER 25 - This should normally be a reasonable starting point. Some users may require more processes, depending on the work they are doing. This value can be set only on a system-wide basis.

MAXPROCSYS 85 - Assume that you need 4 processes per user and 5 processes for daemons. (20 users * 4) + 5 daemons = 85 processes.

Controlling use of ESQA

A number of services use base z/OS functions that use ESQA storage. Much of this storage is fixed, consuming main memory rather than only virtual storage.

Installations having constraints on virtual storage or main memory can control the amount of ESQA storage used by the following services:

- Shared memory
- Memory map files
- ptrace
- fork (copy-on-write)

The following BPXPRMxx parmlib statements are the primary means of controlling consumption by UNIX services:

- MAXSHAREPAGES controls the maximum number of shared pages to be used for fork, shared memory, memory map files, and ptrace. ESQA storage is required for each shared page.
- FORKCOPY determines whether fork should use copy-on-write support. Copy-on-write support should normally reduce the cost of fork by removing the need to copy all the parent's virtual storage to the child address space. However, on systems with storage constraints, the benefit of copy-on-write may be outweighed by the impact on ESQA storage.

Follow these guidelines:

- If the run-time library is in the link pack area, specify FORKCOPY(COPY).
- If the run-time library is not in the link pack area, specify FORKCOPY(COW).

Other statements in the BPXPRMxx parmlib member provide more detailed control of how shared memory, and memory map files can be used.

For more detail on statements in the BPXPRMxx parmlib member, see *z/OS MVS Initialization and Tuning Reference*. For more detail on ESQA and other storage requirements for MVS, see “Evaluating virtual storage needs” on page 17.

Enabling nice(), setpriority(), and chpriority() support

In general, it is not recommended that customers enable nice(), setpriority(), and chpriority() support. Instead, IBM recommends exclusive use of normal SRM controls. However, nice(), setpriority(), and chpriority() support is provided. This support interfaces with SRM and workload manager to provide system control and monitoring support.

If your installation plans to support the **cron** daemon, setpriority() support may be needed. **cron** allows interactive users to schedule work to run in the background at various times in the future. Normally, this background work should run at a lower priority than other interactive work. By default, **cron** uses setpriority() to lower the priority of batch work it starts. The return code is not checked, so if the setpriority() call fails, the batch work simply runs at the same priority as other forked children. This could become a problem if background work started by **cron** begins to affect the responsiveness of foreground interactive work. In this case, it may be appropriate to customize your system to support three levels of dispatching priority (as illustrated in the example below).

How they work

The setpriority() and chpriority() functions let the caller set the dispatching priority for a process, a process group, or a user. The priority value specified can range from -20 to 19. On this scale, -20 is highest priority and 19 is lowest priority. The nice() function allows a calling process to change its own priority.

Resulting nice() values can range from 0 to 39, with 0 being the highest priority and 39 being the lowest. With all three services, appropriate privileges are required to increase the priority of one or more processes.

Priority values (-20 to 19) and nice() values (0 to 39) are mapped one-to-one such that nice() values are always 20 higher than priority values. All processes start with a priority value of 0 and a nice() value of 20.

priority value (setpriority and chpriority)		nice value (nice)
-----		-----
-20	A	0
.	higher priority	.
.		.
0		20
.		.
.	lower priority	.
.		.
.		.
+19		39
	V	

To enable the nice(), setpriority(), and chpriority() functions, an installation must specify a PRIORITYPG statement or PRIORITYGOAL statement in BPXPRMxx. Installations that are running in compatibility mode should use the PRIORITYPG statement to specify a performance group for each possible priority value (-20 to 19). The first value corresponds to a priority value of -20 (very high priority). The next corresponds to a priority value of -19, and so on until the 40th value corresponds with a priority value of 19. If fewer than 40 values are specified, the last value is propagated through the remaining priority values. The same performance group can be specified several times.

Installations that are running in goal mode to exploit MVS workload manager can enable nice(), setpriority(), and chpriority() support using the PRIORITYGOAL statement in the BPXPRMxx parmlib member. They must specify a service class for each possible priority value (-20 to 19). If fewer than 40 service classes are specified, the last service class is propagated to all remaining priority values. The same service class can be specified several times. All service classes specified must appear in your current service policy.

Recommendations

Generally, you should not specify PRIORITYPG and PRIORITYGOAL in BPXPRMxx unless you need nice() and setpriority() support. It is simplest and best to give MVS full control over priorities of work.

Making sure that the sticky bit for the z/OS shell is on

To reduce I/O and improve performance, the z/OS shell is shipped with the sticky bit set on.

Example: To verify that the sticky bit is set on, issue :

```
ls -l /bin/sh
```

Result: The output should be:

```
-rwxr-xr-t
```

The t indicates that the sticky bit is on.

Improving the z/OS shell performance

You can improve the z/OS shell performance by setting the environment variables _BPX_SHAREAS and _BPX_SPAWN_SCRIPT, and by controlling use of STEPLIBs.

Setting environment variables

There are two environment variables you can use to improve performance for the z/OS shell utilities: `_BPX_SHAREAS` and `_BPX_SPAWN_SCRIPT`.

Rule: You cannot use those two environment variables for the tcsh shell.

`_BPX_SHAREAS`

To improve performance in the z/OS shell, set `_BPX_SHAREAS` to YES. The z/OS shell will run foreground processes in the same address space as the shell is running in, which saves the overhead of a `fork()` and `exec()`.

Recommendation: To improve performance for all z/OS shell users, **/etc/profile** or **\$HOME/.profile** should set `BPX_SHAREAS=YES`

```
export _BPX_SHAREAS=YES
```

If you use `_BPX_SHAREAS=YES`, the `spawn()` runs faster, the child process consumes fewer resources, and the system can support more resources.

However, when running multiple processes with `BPX_SHAREAS=YES`, the processes cannot change identity information. For example, `setuid()` and `setgid()` will fail. You cannot execute `setuid()` or `setgid()` in the same address space as another process. Also, when the parent terminates, the child will terminate because it is a subtask

If the extended attribute for the shared address space is not set, the program will not run in a shared address space, regardless of the setting of `_BPX_SHAREAS`. The attribute is set by **extattr +s** and reset by **extattr -s**. If the attribute is set, `_BPX_SHAREAS` has precedence.

`_BPX_SPAWN_SCRIPT`

To improve performance when running the z/OS shell scripts, set `_BPX_SPAWN_SCRIPT` to YES.

The `spawn()` service will run files that are not in the correct format to be either an HFS executable or a REXX exec as shell scripts directly from the `spawn()` function. Because the shell uses `spawn()` to run foreground commands, setting this variable to YES eliminates the additional overhead of the shell invoking `fork` after receiving `ENOEXEC` for an input shell script.

To provide this performance benefit to all shell users, IBM recommends that **/etc/profile** or **\$HOME/.profile** set the environment variable:

```
export _BPX_SPAWN_SCRIPT=YES
```

However, there may be exceptions to this recommendation, depending on your environment.

Avoiding use of STEPLIBs

If you enter the OMVS command either from ISPF or with STEPLIB data sets allocated, include the statements in the shell profile as shown in Figure 55 on page 343 and Figure 56 on page 343.

If you use the OMVS command to login to the shell, you can improve performance by using a logon procedure that does not contain any JOBLIB or STEPLIB DD allocations. This reduces the amount of storage that is copied for `fork()`. It also

prevents excessive searching of STEPLIB data sets and the propagation of STEPLIB data sets from the shell process to the shell command processes on `exec()`.

Statements in either **/etc/profile** or **\$HOME/.profile**, as shown in Figure 55, improve the shell's performance for users who enter the OMVS command from ISPF or with STEPLIB data sets allocated. This prevents excessive searching of STEPLIB data sets and the propagation of STEPLIB data sets from the shell process to the shell command processes on `exec()`. The example in Figure 55 also prevents propagation of STEPLIB data sets to shell processes, which may have been necessary because a specific release level of the Language Environment run-time library is needed.

```
if [ -z "$STEPLIB" ] && tty -s;
then
    export STEPLIB=none
    exec sh -L
fi
```

Figure 55. Eliminating STEPLIB propagation

Exporting specific STEPLIBs

The user may have the Language Environment run-time library (SCEERUN) data set allocated as part of ISPLLIB to invoke OMVS from ISPF. In this case, a subset of the STEPLIB data sets needs to be propagated. To propagate specific STEPLIB data sets, **\$HOME/.profile** is different. Figure 56 is an example of **\$HOME/.profile** so that only the STEPLIB data set CEE.SCEERUN containing the Language Environment run-time library is propagated.

```
if [ -z "$STEPLIB" ] && tty -s;
then
    export STEPLIB=CEE.SCEERUN;
    exec sh -L
fi
```

Figure 56. Propagating only the Language Environment run-time library STEPLIB

A module found in CEE.SCEERUN will be loaded from that library into the user's private storage, even if the same module has been put into the LPA. This can become a concern if the STEPLIB points to a Language Environment run-time library, because several loads are done for each `exec()` to initialize the environment. If you have a number of users accessing this load library, you can avoid directory I/O as well as I/O to load frequently used members by caching the library in LLA and VLF.

Improving performance of file systems

Performance of the HFS file system is highly dependent on how the file system is organized. Because a mountable file system must reside on a single DASD volume, several file systems on a volume or too much activity in a single file system can cause DASD I/O response time to be a bottleneck. In addition, some file system locking is done on a mountable file system basis. For these reasons, each user should normally have a unique mountable file system.

Another consideration is the placement of files in the file system hierarchy. Files deep in the hierarchy require several lookups each time they are opened.

The /tmp directory

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. For more information, see Chapter 16.
- 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.

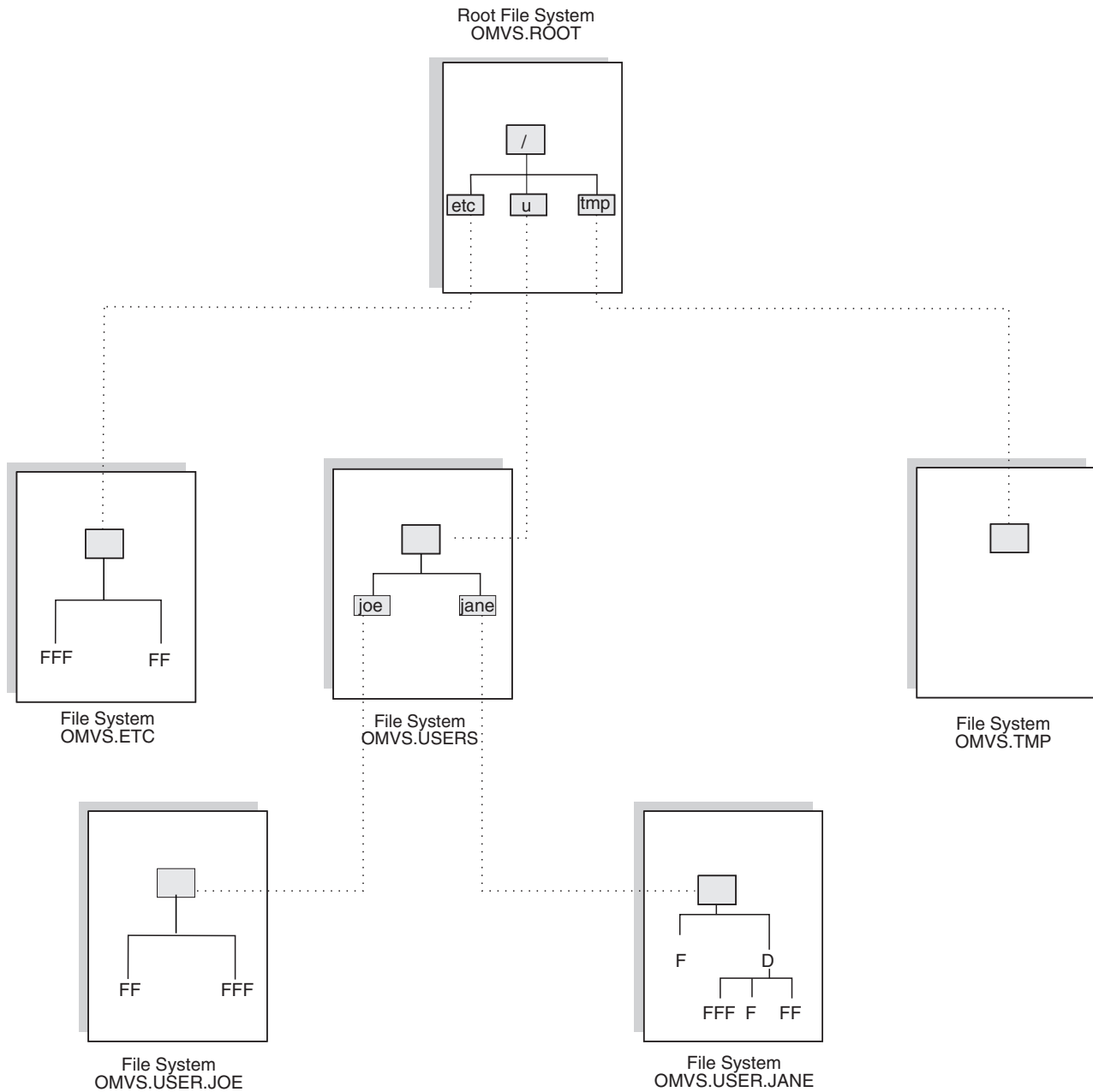


Figure 57. The /tmp directory on a mountable file system

Caching frequently-read files

Files that are read-only can be cached in virtual storage. These files can include data, message catalogs, scripts, and executable programs. Identify files that may be accessed frequently and then use the **/usr/sbin/filecache** utility in your **/etc/rc** script to cache these files in virtual storage. This virtual storage is in a data space associated with the kernel address space.

Improving performance of security checking

To improve the performance of security checking done for z/OS UNIX, define the BPX.SAFFASTPATH FACILITY class profile. This reduces overhead when doing z/OS UNIX security checks for a wide variety of operations. These include file access checking, IPC access checking, and process ownership checking. For more information about the BPX.SAFFASTPATH profile, see “Enabling the fastpath support for system authorization facility (SAF)” on page 281.

OMVS command and TSO/E response time

When a user goes into the shell environment using the OMVS command from TSO/E, very long TSO/E response times (several seconds) may be recorded. This can affect those WLM goals for TSO users that are based on response time.

Normally, a TSO/E transaction starts when a user enters a command and ends when the command is completed. After the TSO/E command completes, a TGET WAIT is issued, indicating that the current transaction has completed and a new transaction will start when there is more work to be done.

In the OMVS shell environment, however, things work a little differently. A transaction starts when a command is issued from the terminal. After the command is issued, polling is done to wait for output to return from the command. Every half second, there is a test for output and a test (TGET NOWAIT) for terminal input. This goes on for 20 seconds before the session goes into INPUT mode and does a TGET WAIT for terminal input only. TGET NOWAIT does not end the current transaction unless terminal input is found. If there is no more terminal input for over 20 seconds, the transaction does not end until the TGET WAIT is issued and the session goes into INPUT mode.

In effect, TSO/E users in the shell environment can experience response times of up to 20 seconds, often with little service consumption. Response times under 20 seconds occur only when users immediately enter the next command.

Improving the performance of the make utility

The z/OS shell users may improve the performance of the **make** utility by setting the **_MAKE_BI** environment variable to YES. The built-in versions of the **make** and **c89/cc/c++** utilities will be used. The **_MAKE_BI** environment variable can be exported and set in **/etc/profile**, **\$HOME/.profile**, or on the command line.

Chapter 21. Setting up for sockets

Overview

When setting up for sockets, your two choices are INET and CINET (Common INET). INET and CINET are file systems that are in the AF_INET and AF_INET6 family of sockets. This chapter helps you decide which file system is best for you to use and describes how to set it up. It contains examples that are based on an assumed sample configuration. You will need to modify the examples based on the requirements for your installation.

User-written socket applications can use TCP/IP as a communication vehicle. TCP/IP is also the transport provider when users rlogin or telnet from a UNIX workstation directly into the z/OS shell.

You can use CINET configured with just one stack, but this configuration will not run as efficiently as INET. “Choosing between INET or CINET” on page 348 provides background information that you may need.

Figure 58 shows an example of a z/OS UNIX system that uses a single stack.

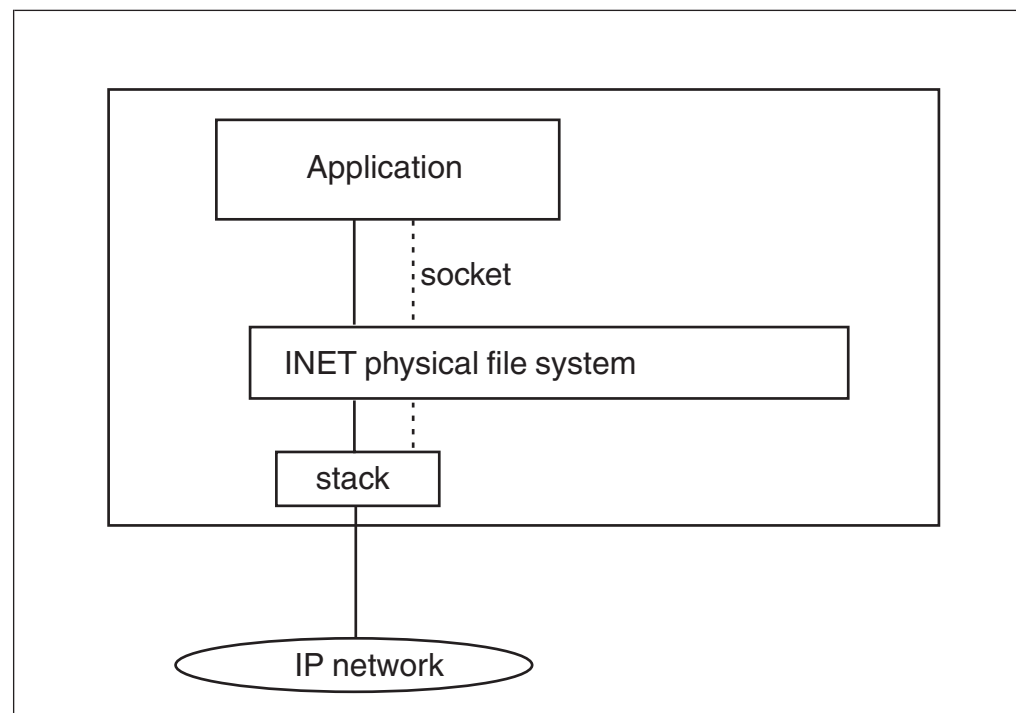


Figure 58. z/OS System using a single stack

The TYPE(INET) parameter on the FILESYSTYPE statement defines INET.

INET is defined by the presence of a FILESYSTYPE statement for a socket file system whose ENTRYPOINT is not BPXTCINT. By convention, and in this chapter, TYPE(INET) parameter for INET configurations is specified. The entry point for INET is usually EZBPFINI, for z/OS Communications Services (TCP/IP Services).

Figure 59 on page 348 shows an example of a z/OS UNIX system that uses multiple stacks.

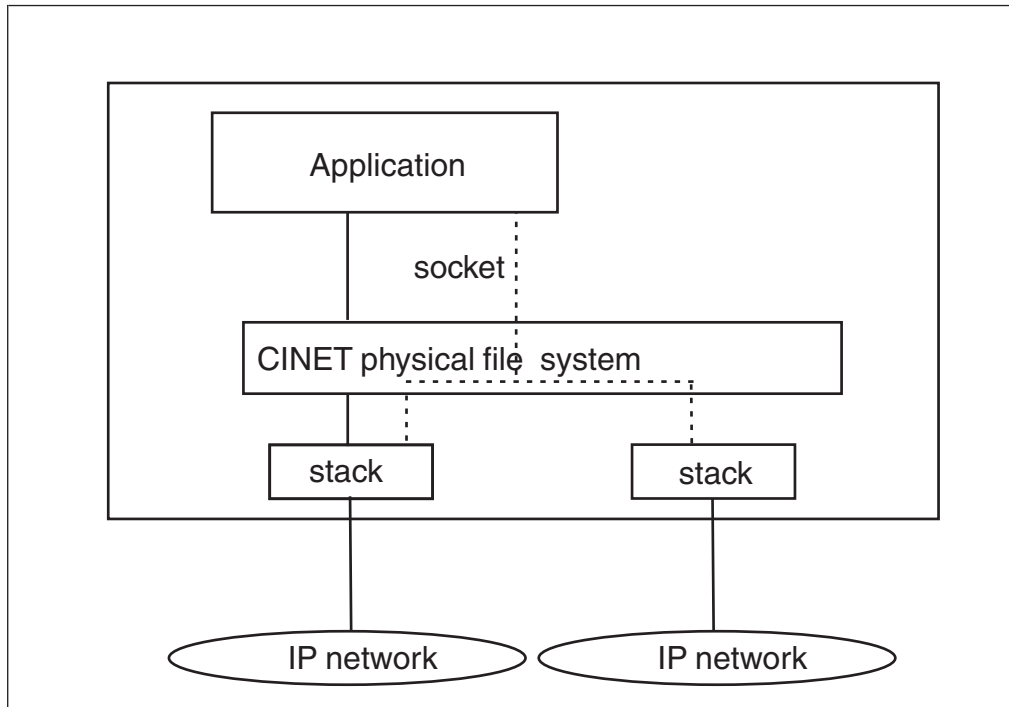


Figure 59. z/OS UNIX system using multiple stacks

CINET is defined by the presence of a FILESYSTYPE statement with an ENTRYPOINT of BPXTCINT and the presence of one or more SUBFILESYSTYPE statements that define the actual TCP/IP stacks to be used. By convention, and in this chapter, TYPE(CINET) parameter for CINET configurations is used. The entrypoints for the SUBFILESYSTYPE statements are usually EZBPFINI for several instances of the z/OS Communications Services (TCP/IP Services), and possibly ISTOEPIT for SNA Communications or the entry point for a conforming TCP/IP stack that is provided by a vendor.

For the example in Figure 59, you would have the two SUBFILESYSTYPE statements to define the two stacks shown in the illustration.

Choosing between INET or CINET

Recent enhancements to TCP/IP have reduced the need for multiple TCP/IP stacks. CINET may still be a viable choice if you are running Anynet and TCP/IP concurrently on a z/OS system or isolating access from different networks to the same z/OS system (internal company networks versus internet access is an example of this situation).

Both this document and *z/OS Communications Server: IP Configuration Guide* contain information on running one TCP/IP stack and multiple TCP/IP stacks

Local INET (LINET) has been retired. SUBFILESYSTYPE ENTRYPOINT(BPXTLINT) was originally supplied as an alternative stack under CINET for a performance enhancement for AF_INET socket sessions between programs on the same system. The performance improvements that have been made in the TCP/IP socket stack since OS/390 V2R5 have removed the need for LINET. If LINET is started, an informational message is sent to the system console

and LINET will deactivate itself. This will not affect the use of sockets, and you can delete the definition of LINET from your BPXPRMxx member at your convenience.

Setting up for INET

Use INET unless you have a special reason to use CINET.

Customizing BPXPRMxx for INET

To use the single transport driver support, see the following example of the statements that should be in the BPXPRMxx parmlib member.

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

Recommendation: To use the single transport driver support, change the MAXSOCKETS value to 64000.

IBM Communications Server for z/OS supports the AF_INET6 address family, which allows socket applications to use the IPv6 APIs. See *z/OS Communications Server: IPv6 Network and Application Design Guide* for more information about IPv6 APIs.

If you want to use the single transport driver support with both AF_INET and AF_INET6 address families, the following excerpt shows an example of the statements that should be in the BPXPRMxx parmlib member:

```
FILESYSTYPE TYPE(INET)
                ENTRYPOINT(EZBPFINI)
NETWORK DOMAINNAME(AF_INET)
                DOMAINNUMBER(2)
                MAXSOCKETS(2000)
                TYPE(INET)
NETWORK DOMAINNAME(AF_INET6)
                DOMAINNUMBER(19)
                MAXSOCKETS(2000)
                TYPE(INET)
```

Tip: You can activate AF_INET6 without recycling z/OS UNIX by adding this NETWORK statement to a running configuration with the SETOMVS RESET=() operator command. Specify a BPXPRMxx parmlib member that contains just this one statement. However, the TCP/IP stack would have to be stopped and restarted in order to pick up the new definition. You can specify a separate MAXSOCKETS value for AF_INET6 or default to the value specified for AF_INET. In either case, each family has its own separate maximum.

Setting up for CINET

The CINET support enables an installation to connect up to 32 transport drivers to z/OS UNIX. The user of the sockets library does not need to change any code to take advantage of the multiple transports connected to the kernel services.

z/OS Communications Server: IP Configuration Reference contains the sample TCP/IP configuration files.

Figure 60 on page 350 and Figure 61 on page 351 show two examples of configurations. In these figures:

- IUTSAMEH is a same-host IP connection and the CTC is channel-to-channel.
- OrouteD and OMPRoute are started dynamic route daemons.
- TCP/IP is shown as the AF_INET socket transport; any conforming socket provider could be used instead.

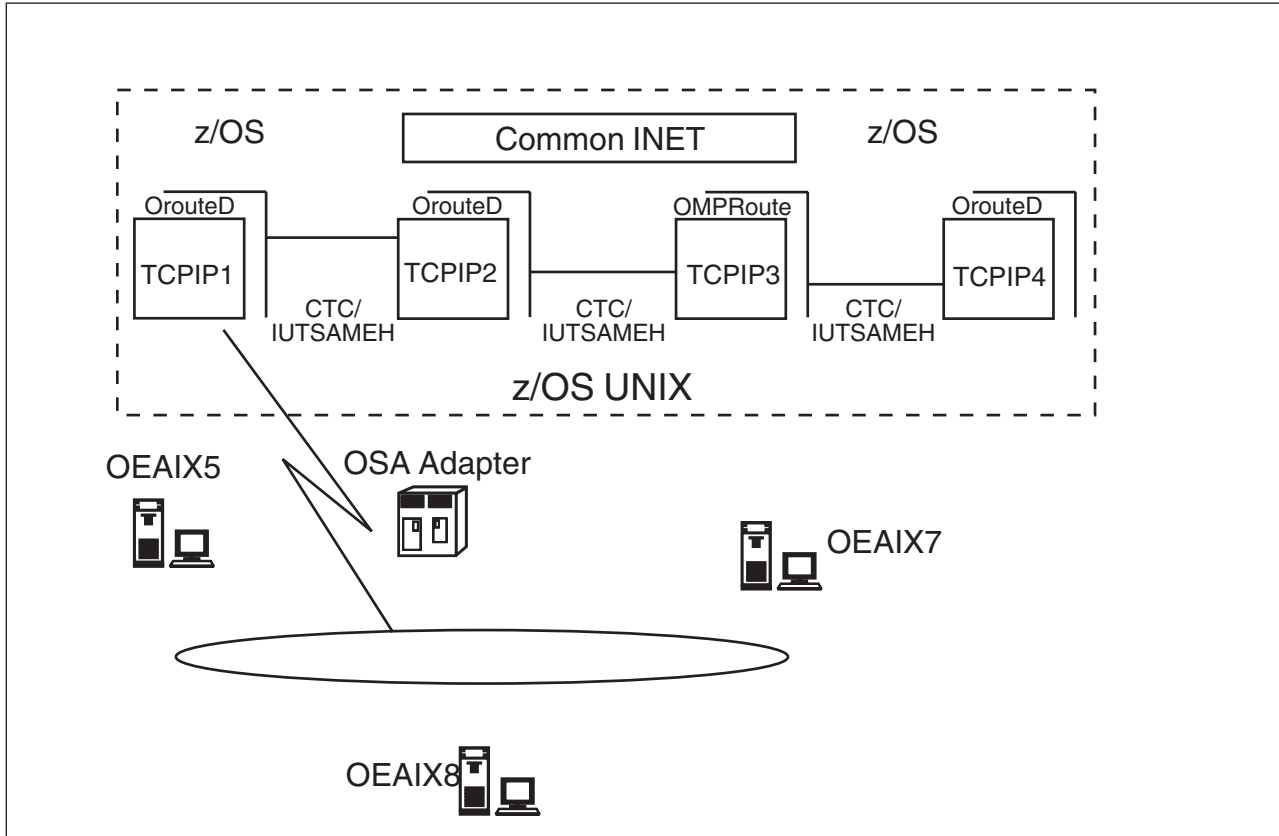


Figure 60. Multiple transport driver support

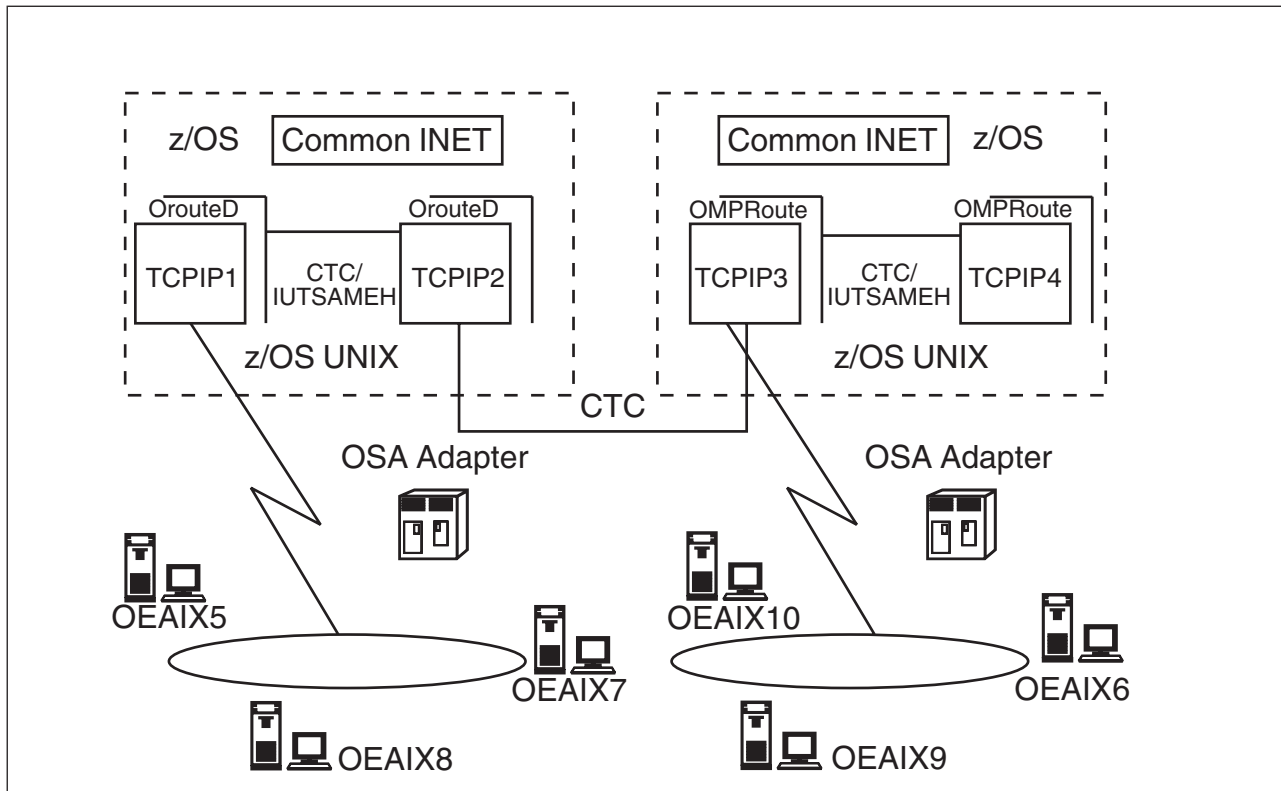


Figure 61. Multiple transport driver support with two z/OS UNIX systems

Supporting multiple transports and providing a single AF_INET or AF_INET6 image to the user means that CINET must perform a set of management and distribution functions that govern how a socket behaves with multiple transports. A fundamental requirement for distributing work across multiple transports is the need to understand the IP configurations of each. The IP configurations are needed to determine which transport should handle a bind(), a connect(), or a sendto() to a particular Internet Protocol (IP) address. An IPv4 address is a 32-bit address defined by the Internet protocols, and an IPv6 address is a 128-bit address defined by the Internet protocols.

When the CINET function processes a socket request that requires it to select only a particular transport based on an input IP address from a user, CINET uses its copy of each transport's IP configuration to select the correct transport to process the user's request. Copies of the IP configurations are maintained by CINET internally and are only used for prerouting a socket call to the correct transport. The transport that is selected performs all of the official transport functions, such as IP routing, once the socket request reaches the transport from CINET.

The internal routing table

Each transport connected to kernel services must provide CINET with a copy of its internal IP routing table. The CINET function queries the routing tables of the transports connected to the kernel services. After the CINET prerouter function has successfully retrieved and stored routing information from a particular transport, message BPXF206I is issued.

For example, IBM's TCP/IP may refresh its routing tables as part of the OBEYFILE command. Message BPXF207I is issued to the hardcopy log whenever CINET

deletes internal routing information for a transport. For example, when a z/OS UNIX-to-TCP/IP connection is severed, the CINET routing information for that TCP/IP is deleted.

You can display the network routing information for all the active transport providers being used by CINET prerouter by using the CINET operand of the DISPLAY OMVS operator command. For example:

```
D OMVS,CINET=ALL
```

Transport drivers

The transport providers were specified with the SUBFILESYSTYPE statements in BPXPRMxx or specified with the SETOMVS command. The default transport driver is one of the following:

- The transport driver specified as the default on the SUBFILESYSTYPE statement in BPXPRMxx.
- The first transport driver that was activated if DEFAULT was not specified or if the specified default transport driver is not active.

Limitations of IP configurations using CINET

System programmers and network designers should be aware of the following information about the CINET prerouting function:

1. **Home IP addresses.** Two or more transports running on z/OS that connect to z/OS UNIX may contain home IP addresses on the same network or subnetwork. However, load balancing across transports is not done.
2. **Network destinations.** Two or more transports may have network destinations that are the same. Again, load balancing across transports is not performed.
3. **Metrics for network routes.** All routes are equal and their metrics are compared.

If two or more transports maintain network routes to the same destination network, metric information is needed from each transport in order to correctly select the best route. For IBM's TCP/IP, this is best accomplished when each TCP/IP is running with a dynamic routing daemon (OrouteD or OMPROUTE). When two or more transports maintain indirect routes to the network, statically defined indirect routes (routes to destinations that do not reside on a transport's directly attached links) do not provide adequate metric information to select the shortest route to a destination network.
4. If two or more transports contain network routes with no metric information or duplicate metrics, then the default transport is called to process the request. The default transport is either the file system that specified DEFAULT on the SUBFILESYSTYPE statement (if active), or it is the first transport that was activated.
5. **Host routes.** Host-defined routes are always searched before network routes.
6. **Severed connection to z/OS UNIX services.** If a transport should sever its connection with z/OS UNIX, all routing information for the severed transport is deleted. If the severed transport maintained duplicate home or network routes, these routes are deleted. Subsequent requests for the duplicate routes are routed to the remaining transports.

Customizing BPXPRMxx for CINET

"Steps for customizing BPXPRMxx for CINET" on page 353 shows an example of the statements in the BPXPRMxx parmlib member to use the multiple transport driver support.

Steps for customizing BPXPRMxx for CINET

Perform the following steps to customize BPXPRMxx for CINET.

1. Specify the following in BPXPRMxx:

```
FILESYSTYPE TYPE(CINET)
            ENTRYPOINT(BPXCINT)
```

2. Specify the AF_INET or dual AF_INET/AF_INET6 sockets physical file systems that are to be activated.

```
SUBFILESYSTYPE NAME(TCPIP1)           /* First TCPIP (TCPIP1) */
                TYPE(CINET)
                ENTRYPOINT(EZBPFINI)
                DEFAULT

SUBFILESYSTYPE NAME(TCPIP2)           /* Second TCPIP (TCPIP2) */
                TYPE(CINET)
                ENTRYPOINT(EZBPFINI)

SUBFILESYSTYPE NAME(TCPIP3)           /* Third TCPIP (TCPIP3) */
                TYPE(CINET)
                ENTRYPOINT(EZBPFINI)

SUBFILESYSTYPE NAME(TCPIP4)           /* Fourth TCPIP (TCPIP4) */
                TYPE(CINET)
                ENTRYPOINT(EZBPFINI)

NETWORK DOMAINNAME(AF_INET) DOMAINNUMBER(2) MAXSOCKETS(2000)
        TYPE(CINET) INADDRANYPORT(4901) INADDRANYCOUNT(100)
```

When you are done, you have customized BPXPRMxx for CINET sockets processing.

Recommendation: Change the MAXSOCKETS value to 64000.

Notes:

1. The names TCPIP1, TCPIP2, TCPIP3, and TCPIP4 are the names of the TCP/IP started tasks. The names must match the jobnames that are associated with the TCP/IP started task procedure.
2. The first TCP/IP has been designated as the default (DEFAULT) transport driver.
3. The value specified for the TYPE operand can be any 8-character value, but that value must match on the FILESYSTYPE statement for CINET, on the SUBFILESYSTYPE statements for the transport drivers, and on the NETWORK statement for CINET.
4. If you want to use IPv6, add the following NETWORK statement:

```
NETWORK TYPE(CINET) DOMAINNAME(AF_INET6) DOMAINNUMBER(19)
```

IPv6 support is optional.

5. For AF_INET6, you can specify a separate MAXSOCKETS value or let it default to the value specified for AF_INET. In either case, each family has its own separate maximum.
6. For AF_INET6, the INADDRANYPORT and INADDRANYCOUNT keywords are ignored. You can activate AF_INET6 without recycling z/OS UNIX by adding this NETWORK statement to a running configuration with the SETOMVS RESET=() operator command. You would specify a BPXPRMxx parmlib member that

contains just this one statement. However, you would have to stop and restart each TCP/IP stack in order to pick up the new definition.

After the default transport driver is assigned, the following actions are taken on sockets calls unless Transport Affinity has been established (see “Transport drivers” on page 352):

- **getsockname()**: If there is a single transport driver that has been connected or bound to, that transport driver is used. Otherwise, the default transport driver is used.
- **gethostname()** or **gethostid()**: The request is always to be routed to the default transport driver.
- **getsockopt()** or **setsockopt()**: If there is a single transport driver that has been connected or bound to, that transport driver is used. Otherwise, the default transport driver is used.
- **Route selection**: When an application program makes a request that could be sent to any of a number of transport drivers—for example, a datagram `sendto()` request—the CINET prerouter examines its internal routing table information and decides which transport driver to send the request to. If more than one transport driver maintains a route to the destination and the routing metric for each of the routes is the same, then the transport driver specified as the default will be chosen.
- **When looking for default routes**, the default transport driver is checked first. If there are no default routes defined for the default transport driver, then the other transport drivers are checked for default routes.

Specifying INADDRANYPORT and INADDRANYCOUNT

These parameters only apply to CINET.

Port reservation information for port 0, INADDR_ANY binds is required for the AF_INET domain with a CINET configuration. Specify this information on the INADDRANYPORT and INADDRANYCOUNT parameters on the NETWORK statement for AF_INET in BPXPRMxx. This also includes the port 0, IN6ADDR_ANY binds for AF_INET6.

If you omit both INADDRANYPORT and INADDRANYCOUNT, then those values will be defaulted. Be careful when allowing the default values because they may not be the values you want. If you do not want to support any reserved ports, then specify INADDRANYPORT(xxx) without specifying INADDRANYCOUNT. In this case, xxx can be any valid numeric value.

INADDRANYPORT specifies the starting port number to be reserved for use by application programs that issue port 0, INADDR_ANY binds. INADDRANYCOUNT specifies how many ports to reserve.

If you are running a CINET configuration and you specify the INADDRANYPORT and INADDRANYCOUNT parameters, you must specify the same values to each transport provider that is specified on the SUBFILESYSTYPE statement.

Refer to the documentation for that transport provider to determine how the port reservation information is specified. For IBM's z/OS Communications Server, use the PORTRANGE profile statement.

If the transport provider does not support the port reservation requirement, you must still specify INADDRANYPORT and INADDRANYCOUNT to process port 0, INADDR_ANY binds. In this case, you should specify a high port number for

INADDRANYPORT (for example, 4000) to improve the probability that the port will be available on the transport provider. If the port is not available on any of the transport providers connected to z/OS UNIX, a port 0, INADDR_ANY bind will fail with an ERRNO of EADDRINUSE.

Using specific transports under CINET

The CINET layer performs a multiplexing/demultiplexing function between an application program and the several transports that are active. When a socket is initially created with the `socket()` call, it is generally available to all the transports. Once the socket becomes associated with a single transport, all subsequent calls go to that one transport; the other transports have no knowledge of the socket at all. Server sockets usually remain associated with all the transports while client sockets often become associated with just one.

The following sections describe the situations that will restrict a socket to just one transport. This is called Transport Affinity.

Binding to a specific transport

Each transport under CINET has its own home IP addresses. When a program binds a socket to a specific IP address, that socket becomes associated with the one transport that supports that IP address.

When a program binds to INADDR_ANY or IN6ADDR_ANY, or an IP address of all zeros, the socket remains available to all the transports. This is also true for sockets that are never bound.

Connecting through a specific transport

When a stream, or TCP, socket is connected, it becomes associated with the single transport that is chosen with the best route to the destination IP address specified on the `connect()` call.

Sockets created from `accept()` are associated with just the one transport on which the connection arrived.

Requesting transport affinity

A program can associate a socket with a specifically-named transport in one of these ways:

1. With **setibmopt(IBM TCP_IMAGE)**, all future `socket()` calls for AF_INET or AF_INET6 create sockets that are associated with only the one specified transport. This can be invoked from non-C programs via **BPX1PCT(PC#SetIbmOptCmd)**. This specification of a specific transport is inherited over `fork()` and propagated over `exec()`.

SetIbmOpt can be issued more than once to change the chosen transport and affect future sockets that are created. If a blank transport name is used, the process is reset so that no transports are chosen.

When CINET is not configured, there is only one AF_INET or dual AF_INET/AF_INET6 transport, and all `socket()` calls for AF_INET or AF_INET6 create sockets with that transport. In this case, `setibmopt()` has no effect and is ignored.

If the specific transport is not found, `setibmopt()` fails with EIBMBADTCPNAME if CINET is installed, and with ENXIO if it isn't.

For more details, see `setibmopt()` in *z/OS C/C++ Run-Time Library Reference* or `pfsctl (BPX1PCT)` in *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.

2. **Call `ioctl(SIOCSETRTTD)`** associates an existing socket with the one specified transport, removing the others, if any, from the socket.
3. The `_BPXK_SETIBMOPT_TRANSPORT` environment variable can be set to the name of the desired transport before starting the program. This variable can also be set in the `PARM=` parameter of an MVS started procedure to have the Language Environment run-time initialization issue a `setibmopt()` call on behalf of the program being started. This variable can also be included in the `_CEE_ENV` file.
4. Include a job step that invokes `BPXTCAFF`. `BPXTCAFF` is invoked as a job step in front of an existing program in a started procedure or submitted job stream. For example:

```
//STEP0 EXEC,PGM=BPXTCAFF,PARM=TPNAME
//REALSTEP EXEC,PGM=MYPGM,PARM='MyParms'
```

The desired transport is specified with the `PARM=` keyword and must be 1 to 8 uppercase characters. This is the same value that would be specified for `_BPXK_SETIBMOPT_TRANSPORT`. If `PARM=` is not supplied, or is blank, then the address space's transport affinity will be reset to no transport selected. This can also be specified as `PARM=&VAR`, where `VAR` is a PROC keyword that is passed in from the Start command or is a static system symbol.

`BPXTCAFF` sets transport affinity for an address space for the duration of that address space or job. This affinity persists over job steps within the job, persists over UNIX process termination and re-dubbing, and applies to all UNIX processes running within that address space. `BPXTCAFF` is intended for use with non-C or POSIX(OFF) programs where the `_BPXK_SETIBMOPT_TRANSPORT` environment variable is not effective. It is also intended for programs that do not make their own calls to `setibmopt()` or `BPX1PCT` or that can not be modified to do so. `BPXTCAFF` exits with one of the following return codes in register 15.

0	Successful. <code>TPNAME</code> matched an <code>AF_INET</code> socket transport
2	Minor failure. <code>TPNAME</code> did not match any transport but <code>CINET</code> is not configured so transport affinity is moot.
8	Failure. <code>CINET</code> is configured and <code>TPNAME</code> did not match any transport running under <code>CINET</code> .
12	Failure. Invalid interface to routine.

To set transport affinity for a TSO address space, you can also invoke `BPXTCAFF` from TSO/E by issuing:

```
TSO CALL 'SYS1.LINKLIB(BPXTCAFF)' 'TPNAME'
```

`BPXTCAFF` makes a call to `BPX1PCT(PC#SetIbmOptCmd)` with an `Arg` value of 1 specified to achieve persistent address space transport affinity. For more details, see `BPX1PCT` in *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.

Rule: A `BPXTCAFF` job step must not be used with z/OS UNIX address spaces that are set up for the NFS or DFS clients. The z/OS UNIX services that are needed by `BPXTCAFF` are not available when the colony is started from the `BPXPRMxx` member but z/OS UNIX cannot finish initialization until the colonies are initialized, so the system will hang up.

5. Use the PARM= parameter of the z/OS UNIX colony address space. To set transport affinity for the NFS Client or DFS Client, use the PARM= keyword of the EXEC statement that starts BPXVCLNY in the colony address space procedure as follows:

```
//MVSCLNT EXEC PGM=BPXVCLNY,TIME=1440,PARM=TP(TPNAME)
```

where the PARM=value is the following:

- All in uppercase
- Starts with "TP("
- TPMNAME is the left-justified, 1-to-8-character name of the desired transport

If PARM= is specified and does not conform to these rules, the colony is terminated by an EC6 abend with a reason code of 11BE8039. When CINET is configured on the system and the specified transport is not configured under CINET, the colony is terminated by an EC6 abend with a reason code of 11BE803A. In either case, the colony can be restarted after the procedure is corrected by replying to the operator prompt that is issued.

Tip: The `_BPXK_SETIBMOPT_TRANSPORT` environment variable does not work in a z/OS UNIX colony address space because it does not start under Language Environment.

Format of resolver configuration data

This section explains the format of the resolver data sets and files when they are stored in the HFS.

Host information

Host information not obtained from a domain name server can be obtained from local host tables. If you want to know what the options are for creating local host tables and how the tables are searched, refer to *z/OS Communications Server: IP Configuration Guide*.

Service information

Figure 62 shows an extract of the services file on z/OS. You can copy the sample service information from `/usr/lpp/tcpip/samples/services` into your **tcpip.ETC.SERVICES** data set or `/etc/services` file. See *z/OS Communications Server: IP Configuration Reference* for more information on the syntax rules for `/etc/services` and ETC.SERVICES.

echo	7/tcp	
echo	7/udp	
discard	9/tcp	sink null
discard	9/udp	sink null
systat	11/tcp	users
daytime	13/tcp	
daytime	13/udp	
netstat	15/tcp	
qotd	17/tcp	
chargen	19/tcp	

Figure 62. Partial extract of the services information

Protocol information

You can copy the sample protocol information from **/usr/lpp/tcpip/samples/protocol** into your tcpip.ETC.PROTO data set or **/etc/protocol** file.

To see the MVS version of the TCPIP.DATA data set, see the sample in the chapter on defining TCP/IP client system parameters in *z/OS Communications Server: IP Configuration Reference*.

Resolver information

The TCPIP.DATA data set is the only one of the TCP/IP data sets for which a unique copy is needed for each transport driver. This is because the TCPIPJOBNAME statement identifies the TCP/IP address space and the HOSTNAME statement identifies the host name of the TCP/IP address space.

The following example shows a typical z/OS TCP/IP syntax:

Dataseprefix	TCPIP.TEST1	; This stack's data set prefix
TCPIPjobname	TCPC01	; Stack name
NSinterAddr	127.0.0.1	; Name server (on this system)
NSportAddr	53	; Name server port number
ResolveVia	UDP	; Use UDP for Name server
ResolverTimeout	30	; 30-second name server timeout;
ResolverUdpRetries	1	; Retry name server once
HostName	TCPIP1	; My host name
DomainOrigin	pok.ibm.com	; My domain origin
Messagecase	mixed	; Issue mixed-case messages

The following example shows a typical z/OS UNIX syntax:

```
domain pok.ibm.com
nameserver 9.114.75.254
nameserver 9.114.171.254
nameserver 9.114.151.254
```

The system processes this information during the initial request for service. It accepts either format for the information supplied regardless of the source selected.

When they are mixed, only the last domain or DOMAINORIGIN data is used and up to 16 name server's addresses are used for initialization. However, if you set up **/etc/resolv.conf** to supply resolver information, you must specify the DATASETPREFIX information in **/etc/resolv.conf** unless you have also set up **/etc/services**, **/etc/protocol**, and **/etc/hosts** files.

Any mix of MVS data sets and HFS files can be used. For example, you could use the TCPIP.DATA information from SYS1.TCPPARMS, the service and protocol information from ETC.SERVICE and ETC.PROTO and use the HFS **/etc** directory to record hosts names and addresses of the HFS hosts file.

Chapter 22. Managing accounting work

Overview

To perform accounting for UNIX workloads, use system management facilities (SMF).

MVS accounting

Your basic accounting models that use address-space level data from SMF record type 30 should work correctly for UNIX processes. Be aware that:

- The TCB time in SMF record type 30 includes the time in the kernel address space.
- The address-space level EXCP (I/O) count includes the I/O for HFS files.
- If the program in a user address space issues `fork()`, the child inherits the TSO/E user ID and the UID.

If you would like to weigh central processor time or I/O or both, use the fields in SMF record type 30 to isolate the resources used. Record type 30 also includes the user identification fields:

- UID
- GID
- Process ID (PID)
- Parent process ID (PPID)
- Process group ID (PGID)
- Session ID (SID)

For detailed file system and file open and close activity data, look in SMF record type 92.

When you perform the accounting, one other major factor to be aware of is that the `exec()` family of functions usually causes step termination and a new substep is started. The new substep still has the same step number, but the substep number is incremented. Therefore, accounting applications must look for `substep_number` in addition to `jobname`, `job_start_time`, and `step_number`.

The kernel creates other address spaces, such as BPXOINIT, and forks other programs—for example, `/etc/init`. The kernel and all its child processes use the same account number. BPXOINIT is the source of the account number propagated to the `/etc/rc` and daemons.

Since the kernel is a started procedure, you can assign accounting data only by coding a JCL installation exit. Alternatively, you can allow the resources used by the kernel and its forked address spaces to be accounted for as system overhead.

For information on SMF records, see *z/OS MVS System Management Facilities (SMF)*.

Assigning account numbers for forked address spaces

Account numbers for forked or spawned address spaces are set as follows:

- Forked or spawned address spaces inherit accounting data from the parent address space.

- When daemon processes such as **rlogind** or **cron** create new work using `setuid()` and `exec()`, accounting data comes from the user's RACF profile (the WAACNT value in the WORKATTR segment). If this value is not defined in RACF, the address space will not have accounting data.
- Accounting data can also be verified or changed using the IEFUAV or IEFUSI installation exits.

Note: Your IEFUSI exit will not receive any step account information for forked address spaces.

- With the `_BPX_ACCT_DATA` environment variable, users can change the account data for a process that is about to be `exec()`d or spawned.
- With the `__spawn()` service, the caller can define account information in the spawn inheritance structure. See *z/OS UNIX System Services Programming: Assembler Callable Services Reference* for more information.

The IEFUAV exit is only passed control when the IEFUAV exit is activated for subsystem OMVS (or all subsystems). This environment typically describes the environment in which a daemon determines the identity of a client, sets up the security environment, and passes the routine control.

In the case of a `fork()`, `spawn()`, or `exec()` where the accounting data is provided by a superuser, the IEFUAV exit is not passed control.

Modifying the accounting information for the OMVS and BPXOINIT address space

To add accounting information for the OMVS or BPXOINIT address spaces, customers must use a function that is documented in *z/OS MVS Initialization and Tuning Reference*.

A summary of the steps is listed below.

1. Change the MSTJCLxx PARMLIB member, so that the JCL contains an IEFJOBS DD statement. This DD statement needs to point to a data set called SYS1.STCJOBS which is FB 80 data set.

This also holds true if you maintain a MSTJCLxx module in SYS1.LINKLIB.

```
//MSTJCL01 JOB MSGLEVEL=(1,1),TIME=1440
//          EXEC PGM=IEEMB860,DPRTY=(15,15)
//STCINRDR DD SYSOUT=(A,INTRDR)
//TSOINRDR DD SYSOUT=(A,INTRDR)
//IEFPDSI  DD DSN=SYS1.PROCLIB,DISP=SHR
//IEFJOBS  DD DSN=SYS1.STCJOBS,DISP=SHR
//IEFPARM  DD DSN=SYS1.PARMLIB,DISP=SHR
//SYSUADS  DD DSN=SYS1.UADS,DISP=SHR
//SYSLBC   DD DSN=SYS1.BROADCAST,DISP=SHR
```

2. In the SYS1.STCJOBS data set, create a member that has the same name as the started procedure; for example, BPXOINIT or OMVS.
3. This member should contain these statements for OMVS:

```
//OMVS JOB (account data),TIME=NOLIMIT,REGION=0K
//OMVS EXEC OMVS
```

This member should contain these statements for BPXOINIT:

```
//BPXOINIT JOB (account data),TIME=NOLIMIT,REGION=0K
//BPXOINIT EXEC BPXOINIT
```

The accounting data for BPXOINIT is propagated to the `/etc/init` process (or `/usr/sbin/init` process) and all the processes they create.

4. The next time the system is IPLed, it uses the updated MSTJCLxx parmlib member. When the OMVS and BPXOINIT address spaces are started, the accounting information will be present.

Account information for TSO users who log into the shell environment and run utilities or shell scripts comes from the TSO/E logon panel account field. This is true even for users who have a WORKATTR segment in the security product data base.

Those users who use rlogin to get into the shell and obtain account information from the WORKATTR field.

IEFUAV — user account validation exit

After the IEFUAV exit receives control for forked/spawned address space, the accounting information can be checked.

If the `_BPX_ACCT_DATA` environment variable and the account data was not specified in the spawn inheritance structure for `_spawn` only, then the account data passed to the exit is the same as the account data of the parent of the forked/spawned address space.

If the `_BPX_ACCT_DATA` environment variable or account data was specified in the spawn inheritance structure for `_spawn`, then this is the account data that is seen in this exit. For spawned address spaces, the account data in the spawn inheritance structure takes precedence over account data from the `_BPX_ACCT_DATA` environment variable.

If the IEFUAV exit sets a return code indicating that the user is not to be allowed to continue, the reason code 0BFC0432 is issued and the address space is terminated.

- 0BFC0432 — issued from module BPXPRJSR
- Reason code 0432 - JRJsrUavXit - The IEFUAV exit rejected the accounting data

If users are running in the shell, they may see the following message:

FSUM9209 cannot execute: reason code 0bfc0432

Message BPXP005I is written to the job log for the user:

BPXP005I A fork or spawn error was encountered.
Return code 00000070 Reason code 0BFC0432

IEFUJI — job initiation exit

If you use the IEFUJI installation exit to check job names, accounting information, or both, you will need to make changes to your OMVS setup.

The OMVS and BPXOINIT address spaces have a subsystem type of STC, SUBSYS(STC). However, the other address spaces that are started by OMVS have a subsystem type of OMVS, SUBSYS(OMVS).

To activate the IEFUJI installation exit for the STC subsystem type, follow these steps:

1. Define OMVS as a new subsystem by adding it to the IEFSSNxx parmlib member.

Example 1: IEFSSNxx in keyword parameter format

```
SUBSYS SUBNAME(OMVS)
```

Example 2: IEFSSNxx in positional parameter format

```
OMVS
```

2. Next, specify this new subsystem type in the SMFPRMxx PARMLIB member.

For example:

```
SUBSYS(OMVS,EXITS(IEFUJI))
```

Specify your installation-specific options for TYPE, INTERVAL, and DETAIL on the SUBSYS statement.

3. If you do not specify IEFUJI in the EXITS option, the IEFUJI exit will not get control for any work attributed to OMVS, such as logging into the shell environment or running utilities or shell scripts.

To check accounting information in the IEFUJI exit, do the necessary steps outlined in the previous section and code the IEFUJI exit appropriately.

If the IEFUJI exit checks the job name in order to meet the installation criteria, you will need to change the IEFUJI exit to ignore the following job names:

```
ETCINIT  
ETCINIT1  
ETCINIT2  
ETCINIT3  
ETCRC
```

You also need to add the names of any daemons you intend to start from **/etc/rc** to this job names exclusion list. Accounting information for the daemons will be propagated from the OMVS address space.

When the IEFUJI exit obtains control of a forked or spawned address space, a flag is set in the interface identifying it as a foreground job. In the past, the only time this flag was set was for TSO address spaces. In a TSO address space, there is a TSB pointed to by ASCBTSB. For an address space of subsystype OMVS, ASCBTSB is zero and no TSB exists. Therefore, you cannot count on having a TSB just because the SMF flag identifies it as a foreground job.

You cannot distinguish a forked address space as being used for foreground or background activity.

If you define SYBSYS(OMVS) in IEFSSNxx and then use SET SMF commands, you may receive a warning message saying that the notification of SYBSYS OMVS failed. Just ignore this message

If the IEFUJI exit sets a return code indication that the user should not be able to continue, the initiator will try again. An attempt is made to fork the address space again and if the IEFUJI exit sets the same return code, then reason code 0BFC0434 is set and the address space is terminated.

- 0BFC0434 — issued from module BPXPRJSR
- Reason code 0434 - JrJsrint - Internal error from BPXPRJSR

If users are running in the shell, they may see the following message:

```
FSUM7726 cannot fork:reason code 0bfc0434
```

Message BPXP005I is written to the job log for the user:

BPXP005I A fork or spawn error was encountered.
Return code 00000070 Reason code 0BFC0434

IEFUJV — job validation exit

The IEFUJV exit is not given control for forked/spawned address spaces. However, the BPXAS job that is used to create the forked/spawned address spaces does go through the IEFUJV exit. The account data that is on the job card for the BPXAS job is propagated from the BPXOINIT job. If a BPXOINIT job is defined in SYS1.STCJOBS and the SYS1.STCJOBS data set is set up so that it is used during IPL, then an installation can define account data there that will be propagated to the BPXAS job.

The BPXOINIT job is started and the account data is saved in SWA blocks in internal text format. Then, later, when the first BPXAS job is started, z/OS UNIX reads the account data that was saved in the SWA blocks and adds the account data to the default job card that it builds. The default job card looks like the following:

```
//BPXYOEJS JOB(acct-data),MSGLEVEL=(0,0),REGION=54M,TIME=60
```

z/OS UNIX takes the account data and reconstructs the internal text to a format that can be placed on a JCL statement. The account data is reconstructed by placing quotes around each account data field and then parentheses around the account data. As a result, the account data may look different than it did when it was defined on the BPXOINIT job. For example, if BPXOINIT had the account data defined as (AA,BB), then when z/OS UNIX adds it to the BPXYOEJS job card, the account data is reconstructed as ('AAA','BBB').

IEFUSI — step initiation exit

When the IEFUSI exit receives control, one of the parameters that is passed is the region size that was requested for the JOB or EXEC testJCL statement. For a forked address space, this is displayed as 54M. This value comes from the default SWA blocks that are defined in module BPXPRBS. This is the default JCL that is used when creating forked address spaces.

If the IEFUSI exit does not change any of the region limit values, then the region value is propagated from parent to child, overriding the 54M value. If the region limit value is changed, then the region value is not propagated. The value set by the IEFUSI exit is used instead.

If the IEFUSI exit sets a return code indication that the user should not be able to continue, the initiator will try again. An attempt is made to fork the address space again and if the IEFUSI exit sets the same return code, then reason code 0BFC0434 is set and the address space is terminated.

- 0BFC0434 — issued from module BPXPRJSR
- Reason code 0434 - JrJsrint - Internal error from BPXPRJSR

If users are running in the shell, they may see the following message:

```
FSUM7726 cannot fork - reason code 0bfc0434
```

Message BPXP005I is written to the job log for the user:

```
BPXP005I A fork or spawn error was encountered.  
Return code 00000070 Reason code 0BFC0434
```

Generating jobnames for OMVS address spaces

When the kernel provides an address space for a fork or spawn request, the following rules are used when generating the jobname:

1. For fork and spawn requests that do not involve changing user IDs, the jobname of the child is set to the base jobname with a number from 1 to 9 appended at the end. For example, if you logon to TSO, you will have a jobname that is the same as your user ID (for example, SMORG). The first fork or spawn creates an address space with SMORG1. In this case, the base jobname is SMORG and all children inherit the same base jobname.
Continuing this example, if address space SMORG1 does a fork or spawn, the new child address space will have a jobname of SMORG1 to SMORG9. It is possible to have multiple address spaces with the same jobname running concurrently.
2. If you run a batch job with a jobname that is 8 characters long (such as PAYROLLX), then all child processes created by this job have the same jobname.
3. When you use rlogin to enter the system, the rlogin daemon prompts for your user ID and password. The daemon then validates the caller and performs a `setuid()` followed by an `exec()`. The `setuid()/exec()` combination triggers the kernel to change the jobname of the address space to the user ID. Because rlogin supports 8-character user IDs, any children created by this process will follow the rules defined in 1 and 2, depending on the length of the user ID.
4. If a daemon issues a `spawn()` with user ID, the child address space is assigned a jobname that is the same as the user ID.
5. If a daemon does a `spawn()` or `exec()` with the `_BPX_JOBNAME` environment variable set, the address space gets the requested jobname.

Any time the jobname is changed, the new jobname becomes the base jobname for future children.

Chapter 23. z/OS UNIX System Services Parallel Environment

Overview

Parallel Environment is installed as part of your z/OS system. This chapter describes what must be done before you can use Parallel Environment.

Setting up the partition daemon

The system administrator must configure the **inetd** daemon so it can start the Partition Manager daemon (**pmd**) when requests from a Parallel Environment home node are received.

1. Add the following line to the **/etc/inetd.conf** file on all remote nodes:

```
pmv2 stream tcp nowait OMVSKERN /bin/pmd pmd
```

2. Add the following line to the **/etc/services** file on the home and remote nodes:

```
pmv2 6125/tcp #POE Partition Manager Daemon
```

You can select a different port number, but the number must be the same on all nodes.

Customizing X-Windows resources

Define the following X defaults files for the graphical debugger (**pedb**) and the Program Marker Array (**pmarray**):

```
ln -f -s /samples/pe/pedb/Pedb.ad /usr/lib/X11/app-defaults/Pedb
ln -f -s /samples/pe/marker/PMarray.ad /usr/lib/X11/app-defaults/PMarray
```

Customizing your code page

If your site is using its own translation of message catalogs, you need to configure **inetd** with your settings of the **LANG** and **NLSPATH** environment variables. This setting enables **pmd** to write error messages and warnings to its log files in the defined language. If you keep the default setting, the messages will be in English.

For example, to set up Japanese error messages for **pmd**, do the following (you need superuser authority):

- Create a shell script, such as **/bin/pmd_Ja_JP**:

```
#!/bin/sh
LANG=Ja_JP NLSPATH=/usr/lib/nls/msg/%L/%N /bin/pmd
```

- Ensure that the shell script is executable by **inetd**.
- Add the following entry in **/etc/inetd.conf**.

- Modify the line in the **/etc/inetd.conf** file as follows:

```
pmv2 stream tcp nowait OMVSKERN /bin/pmd_Ja_JP pmd_Ja_JP
```

- Force **inetd** to read the new setting:

```
> kill -HUP <inetd pid>
```

Using Parallel Environment

Before you can run your job, you must have the same user ID and password on the home node and each remote node on which you will be running your parallel application. You cannot run your application as root.

Each user must have an account on all nodes where a job runs. The user name, the user ID, and the password must be the same on all nodes. Also, the user must be a member of the same group on the home and remote nodes.

Setting up the execution environment

Using host list file

You can use a host file to indicate where your program is to run. This file contains TCP/IP host names, IP addresses, or MVS system names. The specified identifier is used for communication with the **inetd** server. That is, it must be able to resolve to an IP address through the TCP/IP Domain Name Servers or the local TCP/IP HOSTS file.

If you want automatic node allocation using WLM, the identifier specified in the host list file must be valid MVS system names, and they must also be known to TCP/IP as TCP names. You can solve the name restriction problem by defining your MVS system names as aliases in the Domain Name Servers or the HOSTS file. This way, you can always use the MVS system names.

Using Workload Manager Resource Affinity Scheduling

Another way of distributing your work across the nodes is to use the Workload Manager Resource Affinity Scheduling. It enables you to define a set of resources, known as a *scheduling environment*, that a job needs in order to execute and to control the state of each resource for each individual system.

If you want to use Resource Affinity Scheduling, the following must be done:

- The system programmer must add a new scheduling environment and resource names.
- The MVS operator must set the states of additional resources.
- You must give the Parallel Environment user read access to the BPX.WLMSEVER FACILITY class:

```
PERMIT BPX.WLMSEVER ACCESS(READ) CLASS(FACILITY) ID(<user>)
```

Using workload manager multisystem enclave support

For Parallel Environment running on a shared HFS, the WLM multisystem enclave can be used. Two prerequisites are necessary:

1. Define the Coupling facility structure SYSZWLM_WORKUNIT. See *z/OS MVS Planning: Workload Management*. The system command DISPLAY WLM can be used to verify that WLM is connected to this structure.
2. The userid of PMD must have read access to the BPX.WLMSEVER facility class. The PMD user ID is specified in **/etc/inetd.conf** (such as OMVSKERN).

Parallel Environment files

When Parallel Environment is installed as part of your z/OS system, these files are added:

/bin	Shell scripts and executables.
mpcc	Compiles and links parallel C applications.
mpCC	Compiles and links parallel C++ applications.
poekill	Ends all tasks of a parallel program

poe Partition Manager. It calls parallel programs from the home node and controls processing

pmd Partition Manager daemon. It starts parallel programs on remote nodes

mcp Copies files to multiple nodes.

mcpgather
Gathers files from multiple nodes

mcpscat
Scatters files to multiple nodes

pmarray
Starts Program Marker Array

pdbx Parallel command line debugger

pedb Parallel graphical debugger

dbe Debugging engine for pedb

pdbx Parallel debugger for pedb

/usr/lib/

Library services

ppe.dll
Parallel Environment subroutines

ppe.x Exports Parallel Environment subroutines

libppe.a
Parallel Environment subroutines

pe_CEEBINT.o
User exit for Message Parsing Interface initialization

pe_CEEBXITA.o
User exit for Message Parsing Interface initialization

/usr/include

Include files

mpi.h Message Parsing Interface prototypes

mp_market.h
Program Marker Array prototypes

pm_util.h
Parallel Operating Environment utility prototypes

pm_SSM.h
Prototypes for communication subsystems

/usr/lib/nls/msg/C

Message catalogs (English)

pempl.cat
Message Parsing Interface messages

pepoe.cat
Parallel Operating Environment messages

pdbx.cat
Command line debugger messages

pedb.cat
Graphical debugger messages

mpci_err.cat
Subsystem error messages

/usr/lib/nls/msg/Ja_JP/

Message catalogs (Japanese)

pempl.cat

Message catalog for Message Parsing Interface

pepoe.cat

Parallel Operating Environment

pdbx.cat

Command line debugger messages

mpci_err.cat

Subsystem error messages

/samples/pe/marker/

Program Marker Array sources

/samples/pe/pedb/

Resource files and bitmaps for graphical debugger pedb

/usr/man/C/man1/

Man pages (English)

ipeou01.book

Command man pages

/usr/man/Ja-JP/man1/

Man pages (Japanese)

ipeou01.book

Command man pages

Appendix A. Commonly used environment variables

Table 19 summarizes the environment variables that are often used by the kernel.

Table 19. Environment variables often used by the kernel

Environment variable	Purpose
_BPX_ACCT_DATA	Used by the exec() callable service to change the account data of the new process image. For the rules on specifying account data, see the description of the exec() callable service in <i>z/OS UNIX System Services Programming: Assembler Callable Services Reference</i> .
_BPX_JOBNAME	Used by the exec() callable service to change the jobname of the new process image. The jobname change is allowed only if the invoker has appropriate privileges and is running in a space created by fork(). If these conditions are not met, the environment variable is ignored. You can specify a string of 1- to 8 alphanumeric characters. Incorrect specifications are ignored. Appropriate privileges for setting up the jobname includes either superuser authority or READ permission to the BPX.JOBNAME FACILITY class profile.
_BPX_SHAREAS	Used by the spawn() callable service when creating child processes. You can set it to one of the following values: • NO specifies that the child process is to be created in a new address space. This is the default. • YES specifies that the child process is to be created on a subtask in the parent's address space. If the request cannot be honored, the child is created in another address space. • 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 not complete. Sometimes the YES and MUST values cannot be used. See the description of the spawn() callable service in <i>z/OS UNIX System Services Programming: Assembler Callable Services Reference</i>. For information on the importance of _BPX_SHAREAS=YES to performance, see "Improving the z/OS shell performance" on page 341. See "_BPX_SHAREAS" on page 342 for a discussion on the benefits and side effects of using BPX_SHAREAS.
_BPX_SPAWN_SCRIPT	Used by the spawn() callable service to indicate whether the specified file should be treated as a shell script. You can set it to one of these values: • YES specifies that if the specified file is not an executable process image file or a REXX exec, it should be treated as a shell script. Also, the shell is executed to run the specified shell script. The path name for the shell can be specified in the SHELL environment variable on spawn's environment data list, or defaulted to /bin/sh. The first argument passed when spawning a shell script should be the path name of the shell script. • NO indicates that if the specified file is not an executable process image file or a REXX exec, the spawn() service is to fail with a return code of ENOEXEC. This is the default behavior for the spawn() service if _BPX_SPAWN_SCRIPT is not specified or if it contains an unsupported value. For information on the importance of _BPX_SPAWN_SCRIPT=YES to performance, see "Improving the z/OS shell performance" on page 341.

Table 19. Environment variables often used by the kernel (continued)

Environment variable	Purpose
_BPX_USERID	Used by the spawn() callable service to indicate that the child process should be created with the specified MVS user identity. For example, _BPX_USERID=DANIEL will create a child process with the DANIEL user ID to run the spawned program. Authorization for use of the _BPX_USERID is the same as that for the setuid() function. Child processes running with a different user identity than the parent's are always created in a new address space. _BPX_SHAREAS is ignored in this case. Using _BPX_USERID can improve performance for a program that otherwise would have forked a new address space, then established a new user identity for the new address space by issuing initgroups(), setgid(), and setuid(), and so forth, followed by an exec() of the program that is to run under the new user identity. With _BPX_USERID, a program can establish a new user identity for the child that runs the spawned program, rather than creating a child with the old user identity and having the child establish a new user identity. Be careful when using _BPX_USERID. Typically, environment variables passed along on spawn() are still active in the child process. If _BPX_USERID is set in the parent and not cleared in the child, any spawn() calls issued by the child picks up the same _BPX_USERID setting. This is likely to be undesirable. The support that allows the specification of a user ID in the inheritance structure on spawn() does not have this drawback.
_BPXK_INET_FASTPATH	Used by TCP/IP. See <i>z/OS Communications Server: IP Configuration Guide</i> for more information.

Table 19. Environment variables often used by the kernel (continued)

Environment variable	Purpose
_BPXK_JOBLOG	Specifies that WTO messages are to be written to an open HFS job log file. You can set it to one of these values: • <i>nn</i> specifies that job log messages are to be written to open file descriptor <i>nn</i>. • NONE specifies that job log messages are not to be written. This is the default. • STDERR specifies that messages are to be written to the standard error file descriptor, 2. The file that is used to capture messages can be changed at any time by calling the <code>oe_env_np</code> (BPX1ENV) service and specifying <code>_BPXK_JOBLOG</code> with a different file descriptor. Message capturing is turned off if the specified file descriptor is marked for close on a <code>fork()</code> or <code>exec()</code>. Message capturing is process-related. All threads under a given process share the same job log file. Message capturing can be initiated by any thread under that process. Multiple processes in a single address space can each have different files active as the JOBLOG file; some or all of them can share the same file, and some processes can have message capturing active while others do not. When the file that is used as a job log is shared by several processes (for example, by a parent and child), the file should be opened for append. If the file is not opened, unpredictable results may occur. Only files that can be represented by file descriptors can be used as job log files; MVS data sets are not supported. Message capturing is propagated on a <code>fork()</code> or <code>spawn()</code>. If a file descriptor was specified, the physical file must be the same before message capturing can continue in the forked or spawned process. If STDERR was specified, the file descriptor can be remapped to a different physical file. You can override message capturing on <code>exec()</code> or <code>spawn()</code> by specifying the <code>_BPXK_JOBLOG</code> environment variable as a parameter to the <code>exec()</code> or <code>spawn()</code>. Message capturing only works in forked (BPXAS) address spaces. This is not true job log support. Messages that would normally go to the JESYSMSG data set are captured, but messages that go to JESMSGGLG are not captured.
_BPXK_MDUMP	Specifies where a SYSMDUMP is to be written to. You can set it to one of these values: • OFF specifies that the dump is to be written to the current working directory. This is the default. • MVS data set name specifies that the dump is to be written to an MVS data set. The data set name must be a fully qualified name and can be up to 44 characters long. The name is folded to uppercase if typed in lowercase or mixed case. It must also be preallocated and cataloged. • HFS file name specifies that the dump is to be written to an HFS file. The filename can be up to 1023 characters long. The HFS file name must be an absolute path name; that is, it must begin with a slash. The slash refers to the root directory.
_BPXK_SETIBMOPT_ TRANSPORT	Used in a Common INET configuration to choose a socket stack for a program. See Chapter 21 for more information.
SHELL	Used to specify the path name of the shell executable file. The spawn callable service uses this environment variable; see the description of <code>_BPX_SPAWN_SCRIPT</code> .

Table 19. Environment variables often used by the kernel (continued)

Environment variable	Purpose
STEPLIB	Used by the exec(), attach_exec(), attach_execmvs(), execmvs(), and spawn() callable services to create or propagate a STEPLIB environment to the new process image. You can set it to one of these values: • NONE. A STEPLIB DD is not created for the new process image. • CURRENT. If they are cataloged, the TASKLIB, STEPLIB, or JOBLIB DD data set allocations that are active for the calling task at the time of the system call are propagated to the new process image. Uncataloged data sets are not propagated to the new process image. • STEPLIB=Dsn1:Dsn2:,...DsnN. The specified data sets, Dsn1:Dsn2:...DsnN, are built into a STEPLIB DD in the new process image. The actual name of the DD is not STEPLIB, but is a system-generated name that has the same effect as a STEPLIB DD. The data sets are concatenated in the order specified. Restrictions on the data sets that you can specify can be found in the description of the callable services that use this environment variable.

FOMTLINP module for login function

This module is the interface used by rlogin and telnet in z/OS UNIX. In UNIX, rlogin calls the **login** command; for the z/OS shell, **rlogin** calls this module. For z/OS UNIX, rlogin checks passwords.

373

```

*          catopen() will supply the full path
*          name by looking at any inherited settings
*          for LC_MESSAGES, NLSPATH, etc. (note that
*          catopen() is issued with the NL_CAT_LOCALE
*          parameter.)
*
*
*          -- argv[3] = message catalog set for catopen()
*
*          "0" means to use the default catalog set,
*          which is 2
*
*
*          -- argv[4] = master pseudo-TTY file descriptor
*
*          The correct value is required, if the master
*          TTY file descriptor is open in the
*          spawned process. If the master TTY is
*          closed (perhaps because FD_CLOEXEC was set),
*          this parameter must be the number of some
*          closed file descriptor.
*
*
*          -- argv[5] = slave pseudo-TTY file descriptor
*
*          (correct value is required -- must be open)
*
*
*          -- argv[6] = highest used file descriptor
*
*          (This value is used only if fcntl(F_CLOSEFD)
*          fails (perhaps because one of the file
*          descriptors was opened by an authorized
*          program, etc.). fcntl will then close
*          (one-by-one) all file descriptors from 3 to
*          argv[6] + argv[7] onclusive.)
*
*          This argument should not be needed by anyone
*          other than the TSO/E OMVS command.
*
*
*          -- argv[7] = extra file descriptors to close
*
*          (This value is used only if fcntl(F_CLOSEFD)
*          fails (perhaps because one of the file
*          descriptors was opened by an authorized
*          program, etc.). fcntl will then close
*          (one-by-one) all file descriptors from 3 to
*          argv[6] + argv[7] inclusive.)
*
*          This argument should not be needed by anyone
*          other than the TSO/E OMVS command.
*
*
*          -- argv[8] = debug level
*
*          controls whether or not (hidden) debug
*          messages are sent to the TTY slave file
*          descriptor or STDERR (after it has been
*          set up). These messages contain debug
*          information, but are backspaced over and
*          overwritten with blanks, so they would not
*          usually appear on the screen. They will
*          appear in traces, etc. This option is
*          meant to work in conjunction with TSO/E
*          OMVS command debug mode.

```



```

*
*      0 = don't do any debug recording
*      1 = don't do any debug recording
*      2 = don't do any debug recording
*      3 = don't do any debug recording
*
*      4 = do debug recording, with overwriting
*          to hide message on display screen
*
*      5 = do debug recording, but don't try to
*          overwrite the debug text on the screen
*
*      6 = do debug recording to syslog
*
*
*      -- argv[9] = screen width for debug messages
*
*          This value should be set to the width of
*          the display screen, if the debug level is
*          set to 4. It is used when backspacing and
*          erasing the debug messages.
*
*          This value may be set to 0, if the debug
*          level is not 4.
*
*      31 argv[10] = remote hostname (or null) -- only 15
*          bytes of this will fit into the utmpx
*          entry
*
*
*      255 argv[11] = text for TERM environment variable
*
*          TERM is not set, if this is an empty
*          string ("").
*
*
*      15 argv[12] = text for ROWS environment variable
*
*          ROWS is not set, if this is an empty
*          string ("").
*
*
*      15 argv[13] = text for COLUMNS environment variable
*
*          COLUMNS is not set, if this is an empty
*          string ("").
*
*
*      47 argv[14] = path name for SETUID utmpx recording
*          routine
*
*          Empty string ("") means use the default
*          path, which is "/bin/fomtlinc"
*
*
*      15 argv[15] = program name for SETUID utmpx recording
*          routine
*
*          Empty string ("") means use the default
*          program, which is "fomtlinc"
*
*
*      -- argv[16] = SIGCHLD reset flag

```

```

*           1 = SIGCHLD will be reset to the default
*             handling (This value should seldom
*             (if ever) be needed -- the main
*             purpose in the past was to be sure
*             that NOCLDSTOP was off.)
*
*           0 = SIGCHLD handling will not be changed
*
*
* Other expected input conditions:
* -----
*
* 1) Slave TTY must be open, with no controlling terminal
*     established yet.
*
*
* 2) All signals (except perhaps SIGCHLD) should be in their
*     default handling state, before this routine is called.
*     It is OK for signals to be blocked when this routine is
*     called, however.
*
*
* 3) Any environment variables other than NLSPATH, LC_SYNTAX,
*     LC_MESSAGES, LC_CTYPE, and LC_COLLATE will be passed through
*     to the invoked shell. The environment variables named here
*     will be gotten rid of before the shell is called. They
*     will control message catalog processing before the shell is
*     invoked.
*
*
* 4) Little validity checking is done on the parameters, which
*     are expected to be correct. This command is not designed to
*     be run from the shell command line.
*
*
*
* Return Value: Does not return to caller
* -----
*
*
* Non-returning Exits: Does not return to caller
* -----

```

FOMTLOUT module for logout function

The FOMTLOUT module (/bin/fomtlout) performs the logout function.

```

*****
*
* Function:
* -----
*
* This function uses the utmpx
* functions to locate and remove the caller's USERID from the utmpx
* file. This routine is a SETUID program, so that it can write to the
* utmpx file.
*
* note: This routine removes the caller's session from the utmpx file
* whenever it is called. If this routine is called erroneously
* (when the user is not really logging off) it will go ahead and
* remove the session from the utmpx file. This will destroy the
* integrity of the utmpx file.
*
*

```

```

* Parameters:
* -----
*
* 1:IN      argc -- usual main() parameters
*
* 2:IN      argv -- usual main() parameters
*
*      argv[0] = program name ("fomtlinc")
*
*      argv[1] = exit status (as a %d-coded integer value)
*
*      argv[2] = (not used)
*
*      argv[3] = (not used)
*
*      argv[4] = debug level (as a %d-coded integer = 0 to 5)
*
*      others -- test only arguments follow
*
*
*
*      assumed file descriptors on entry:
*
*      0 -- master TTY (needed for ttyname())
*      1 -- might be a debug fd (debug messages written here)
*      2 -- might be an error fd (for system error messages)
*
*
*
* Return Value:
* -----
*
*      1-byte exit status contains 2 bits of syscall ID and 6 bits of
*      compressed errno information for the parent. This information is
*      used only by the TSO/E OMVS command to put out logoff-oriented
*      error messages to the user's terminal.
*
*
*
* Non-returning Exits: none
* -----
*
*
* Main Side Effects:
* -----
*
*      The caller's session is removed from the utmpx file
*
*
*

```

Appendix C. Accessibility

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

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

Using assistive technologies

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

Keyboard navigation of the user interface

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

z/OS Information

z/OS information is accessible using screen readers with the BookServer/Library Server versions of z/OS books in the Internet library at:

<http://www.ibm.com/servers/eserver/zseries/zos/bkserv/>

One exception is command syntax that is published in railroad track format; screen-readable copies of z/OS books with that syntax information are separately available in HTML zipfile form upon request to mhvrfs@us.ibm.com.

Notices

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

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

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

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

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

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

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

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Mail Station P300
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

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

Programming Interface Information

This book is intended to help the customer plan for, customize, operate, manage, and maintain a z/OS system with z/OS UNIX System Services (z/OS UNIX).

This book primarily documents intended Programming Interfaces that allow the customer to write programs that use z/OS UNIX.

This book also documents information that is NOT intended to be used as Programming Interfaces of z/OS UNIX. This information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking:

———— **NOT Programming Interface information** ————

———— **End of NOT Programming Interface information** ————

Trademarks

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

ACFT/VTAM	MVS
AD/CYCLE	MVS/ESA
AIX	NetView
AIX/ESA	Open Class
AnyNet	OpenEdition
BookManager	OS/2
Bookmaster	OS/390
C/370	Parallel Sysplex
CICS	PR/SM
CICS/ESA	PS/2
Common User Access	RACF
DB2	Resource Link
DFS	RISC System/6000
DFSMS	RMF
DFSMSdfp	SAA
DFSMSdss	S/390
DFSMSHsm	SecureWay
DFSMS/MVS	Sysplex Timer
Encina	System Application Architecture
Hiperspace	System/360
IBM	System/370
IBMLink	Tivoli
IMS	VTAM
IMS/ESA	WebSphere
InfoPrint	z/OS
Language Environment	z/Series
Library Reader	

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

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

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

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

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

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

Glossary

This glossary defines technical terms and abbreviations used in z/OS UNIX documentation. If you do not find the term you are looking for, refer to the index of the appropriate z/OS UNIX manual or view *IBM Glossary of Computing Terms*, located at:

<http://www.ibm.com/ibm/terminology>

This glossary includes terms and definitions from:

- *Information Technology—Portable Operating System Interface (POSIX)*, from the POSIX series of standards for applications and user interfaces to open systems, copyrighted by the Institute of Electrical and Electronics Engineers (IEEE). Copies of all POSIX drafts and standards may be purchased from IEEE at 1-800-678-IEEE.
 - Definitions identified by *[POSIX.0]* are from *Part 0: Standards Project, Draft Guide to the POSIX Open System Environment*, P1003.0 Draft 15 (June 1992), an unapproved draft subject to change.
 - Definitions identified by *[POSIX.1]* are from *Part 1: System Application Program Interface (API) [C Language]*, approved September 28, 1990, as IEEE Std 1003.1-1990 by the IEEE Standards Board, and adopted in 1990 as an International Standard (ISO/IEC 9945-1: 1990) by the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC).
 - Definitions identified by *[POSIX.2]* are from *Part 2: Shell and Utilities*, P1003.2.
- *American National Standard Dictionary for Information Systems*, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies may be purchased from the American National Standards Institute, 11 West 42nd Street, New York, New York 10036. Definitions are identified by the symbol *[A]* after the definition.
- *Information Technology Vocabulary*, developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1). Definitions from published sections of these vocabularies are identified by the symbol *[I]* after the definition. Definitions taken from draft international standards, committee drafts, and working papers being developed by

ISO/IEC JTC1/SC1 are identified by the symbol *[T]* after the definition, indicating that final agreement has not yet been reached among the participating national bodies of SC1.

- *CCITT Sixth Plenary Assembly Orange Book, Terms and Definitions* and working documents published by the International Telecommunication Union, Geneva, 1978. These are identified by the symbol *[CCITT/ITU]* after the definition.
- Open Software Foundation (OSF). These are identified by the symbol *[OSF]* after the definition. Copies of OSF documents may be obtained from Open Software Foundation, Inc., 11 Cambridge Center, Cambridge, MA 02142.

Sequence of Entries: For clarity and consistency of style, this glossary arranges the entries alphabetically on a letter-by-letter basis. In other words, only the letters of the alphabet are used to determine sequence; special characters and spaces between words are ignored.

Organization of Entries: Each entry consists of a single-word or multiple-word term or the abbreviation or acronym for a term, followed by a commentary. A commentary includes one or more items (definitions or references) and is organized as follows:

1. An item number, if the commentary contains two or more items.
2. A usage label, indicating the area of application of the term, for example, "In programming," or "In TCP/IP." Absence of a usage label implies that the term is generally applicable to z/OS UNIX, to IBM, or to data processing.
3. A descriptive phrase, stating the basic meaning of the term. The descriptive phrase is assumed to be preceded by "the term is defined as ...". The part of speech being defined is indicated by the opening words of the descriptive phrase: "To ..." indicates a verb, and "Pertaining to ..." indicates a modifier. Any other wording indicates a noun or noun phrase.
4. Annotative sentences, providing additional or explanatory information.
5. References, directing the reader to other entries or items in the dictionary.

6. A source label—for example, *[A]*, *[I]*, *[T]*, *[CCITT/ITU]*, *[OSF]*, *[POSIX.0]*, *[POSIX.1]*, or *[POSIX.2]*—that follows the definition and identifies the originator of the definition. Definitions without source labels are IBM definitions.

References: The following cross-references are used in this glossary:

Contrast with. This refers to a term that has an opposed or substantively different meaning.

Synonym for. This indicates that the term has the same meaning as a preferred term, which is defined in its proper place in the glossary.

Synonymous with. This is a backward reference from a defined term to all other terms that have the same meaning.

See. This refers you to multiple-word terms that have the same last word.

See also. This refers the reader to related terms that have a related, but not synonymous, meaning.

Deprecated term for or Deprecated abbreviation for. This indicates that the term or abbreviation should not be used. It refers to a preferred term, which is defined in its proper place in the glossary.

Selection of Terms: A term is a word or group of words to be defined. In this glossary, the singular form of the noun and the infinitive form of the verb are the terms most often selected to be defined. If the term may be abbreviated, the abbreviation is given in parentheses immediately following the term. The abbreviation is also defined in its proper place in the glossary.

A

a.out. (1) An output file produced by default for certain commands. By default, this file is executable and contains information for the symbolic debugger. [OSF] (2) The object file format created by the **cc** command and expected by the **exec** system call. [OSF]

abend. (1) Abnormal end of task. (2) Synonym for *abnormal termination*.

abnormal termination. (1) The cessation of processing prior to planned termination. [T] (2) A system failure or operator action that causes a job to end unsuccessfully. (3) Synonymous with *abend*, abnormal end.

abort. Synonym for *stop*.

absolute address. (1) A direct address that identifies a location without reference to a base address. [T] (2)

An address that, without the need for further evaluation, identifies a storage location or a device. [OSF] (3) An address that is permanently assigned by the machine designer to a storage location. [A]

absolute pathname. (1) A pathname beginning with a slash. The predecessor of the first filename in the pathname is taken to be the root directory of the process. [POSIX.1] (2) The name of any directory or file expressed as a string of directories and files beginning with the root directory. (3) The name of a shared library key that begins with a slash. The system does not append any directories to the name of the shared library key, and it attempts to open a file with that name. [OSF] (4) The *full* name of a file, possibly including the name of the device the file is on. (A device name is expressed as a letter followed by a colon, as in **C:**) The topmost directory name may be preceded by a slash, and all directory names are followed by a slash. Examples are **c:/dir1/dir2/file.ext** and **/dir/subdir/file.ext**. Synonymous with full pathname. (5) See also *pathname*, *relative pathname*.

absolute value. The numeric value of a real number regardless of its algebraic sign (positive or negative). [OSF]

access. (1) To obtain the use of a computer resource. [T] (2) The manner in which files or data sets are referred to by the computer. (3) The ability to obtain use of a protected resource. [OSF] (4) To obtain data from or to put data in storage. (5) In computer security, a specific type of interaction between a subject and an object that results in the flow of information from one to the other. See also *write access*.

access ACL. An ACL that is used to provide protection for a file system object.

access control. In computer security, ensuring that the resources of a computer system can be accessed only by authorized users in authorized ways.

access control list (ACL). (1) In computer security, a collection of all access rights for one object. In computer security, a list associated with an object that identifies all the subjects that can access the object and their access rights; for example, a list associated with a file that identifies users who can access the file and identifies their access rights to that file. (2) In z/OS UNIX, an extension to the base POSIX permission bits. Similar to the access list of a RACF profile, an ACL for a file system object contains entries that specify access permissions for individual users and groups.

| **ACL.** See *Access control list*.

accessible. Pertaining to an object for which a client has a valid designator or handle.

access method. A technique for moving data between main storage and I/O devices.

access mode. A form of access permitted to a file. [POSIX.1]

access permission. A group of designations that determine who can access a particular file and how the user can access the file.

account. The login directory and other information that gives a user access to the system. See also *user ID*.

ACF/VTAM. Advanced Communications Function for the Virtual Telecommunications Access Method.

action. (1) In *awk*, *lex*, and *yacc*, a C language program fragment that defines what the program does when it recognizes input. [OSF] (2) In SAA Common User Access[®] architecture, a defined task that an application performs. An action modifies the properties of an object or manipulates the object in some way.

address. (1) A character or group of characters that identifies a register, a particular part of storage, or some other data source or destination. [A] (2) A name, label, or number identifying a location in storage, a device in a system or network, or any other data source. (3) An unambiguous name, label, or number that identifies the location of a particular entity or service. [OSF] (4) The telephone number that remote systems use to call the system. [OSF] (5) A value that identifies a register, a particular part of storage, a data source, or a data sink. The value is represented by one or more characters. [OSF] (6) A reference to one or more lines in a file. The simplest address is a line number. More complex addresses can involve regular expressions that search for a particular line. See also *regular expression* (7) To refer to a device or an item of data by its address. [I] (8) In word processing, the location, identified by an address code, of a specific section of the recording medium or storage. [T] (9) In data communication, the unique code assigned to each device or workstation connected to a network.

addressing. (1) In data communication, the way that the sending or control station selects the station to which it is sending data. (2) A means of identifying storage locations. (3) Specifying an address or location within a file. (4) The assignment of addresses to the instructions of a program. See also *selection*.

address space. (1) The memory locations that can be referred to by a process. [POSIX.1] (2) The code, stack, and data that are accessible by a process. (3) The complete range of addresses that is available to a programmer. (4) The area of virtual storage available for a program. (5) See *forked address space*, *kernel address space*, *user address space*.

adjust. To move text in order to make it fit between the left and right margins.

administrator. See *security administrator*.

Advanced Communications Function for the Virtual Telecommunications Access Method (ACF/VTAM).

A licensed program that provides single-domain network capability and, optionally, multiple-domain capability. It controls communication and the flow of data in an SNA network between terminals and application programs running under VSE and OS/VS2.

advisory lock. A type of lock that a process holds on a region of a file preventing any other process from locking the region or an overlapping region. See also *enforced lock*.

aggregate. In programming languages, a structured collection of data objects that form a data type. [I]

alert. (1) To cause the user's terminal to give some audible or visual indication that an error or some other event has occurred. When the standard output is directed to the terminal device, the method of alerting the terminal user is unspecified. When the standard output is not directed to the terminal device, the alert shall be accomplished by writing the <alert> character to standard output (unless the utility description indicates that the use of standard output produces undefined results in this case). [POSIX.2] (2) A character in the output stream that indicates that a terminal should alert its user via a visual or audible notification. The <alert> is the character designated by "\a" in the C language binding. It is unspecified whether this character is the exact sequence transmitted to an output device by the system to accomplish the alert function. [POSIX.2]

ALET. A token that indexes into an access list. An ALET indicates the address space or data space that the system is to refer to.

algorithm. A finite set of well-defined rules for the solution of a problem in a finite number of steps; for example, a complete specification of a sequence of arithmetic operations for evaluating sine *x* to a given precision. [OSF]

alias. (1) An alternate name for a shell command. An alias for a command can be defined with options different from those for the command itself. An alias can be a convenient shorthand for a complicated command line, such as a pipeline. (2) An alternate name for a user, system, or file that can be used in place of the real name of the object. [OSF] An alias may be defined to change the default options given to a particular command (for example, *pg*), or may be a shorthand for a more complicated command line (for example, a pipeline). (3) An alternate label. For example, a label and one or more aliases may be used to refer to the same data element or point in a computer program. [A] (4) An alternate name for a member of a partitioned data set. (5) Synonymous with nickname.

alias address. An alternative address for a network interface that can be used in place of the real address.

alias user ID. A user name conforming to the XPG4 portable character set that is used to access the system. When logging on to z/OS, the alias name is used to locate a z/OS user ID. All security processing is done with the z/OS user ID.

allocate. To assign a resource, such as a disk file, to a specific task. Contrast with *deallocate*.

alphabetic. Pertaining to the set of letters and symbols, excluding digits, used in a language. This set usually consists of the uppercase and lowercase letters plus special symbols (such as \$ and _) allowed by a particular language. [OSF] See also *alphabetic character*.

alphabetic character. A letter or other symbol, excluding digits, used in a language: usually the uppercase and lowercase letters plus other special symbols (such as \$ and _) allowed by a particular language. See also *alphanumeric character*, *digit*.

alphanumeric. (1) Pertaining to data that consists of letters, digits, and usually other characters, such as punctuation marks. [T] (2) Pertaining to a character set that contains letters, digits, and usually other characters, such as punctuation marks. [A] (3) Synonymous with *alphanumeric*.

alphanumeric character. A letter, number, or other symbol, such as a punctuation mark or mathematical symbol. See also *alphabetic character*, *digit*.

American National Standard Code for Information Interchange (ASCII). The code developed by ANSI for information interchange among data processing systems, data communication systems, and associated equipment. The ASCII character set consists of 7-bit control characters and symbolic characters. [OSF] See also *extended binary-coded decimal interchange code (EBCDIC)*.

American National Standards Institute (ANSI). An organization consisting of producers, consumers, and general-interest groups, that establishes the procedures by which accredited organizations create and maintain voluntary industry standards in the United States. [A]

ANSI. American National Standards Institute.

angle brackets. The characters “<” (*left-angle-bracket*) and “>” (*right-angle-bracket*). When used in the phrase “enclosed in angle brackets” the symbol “<” immediately precedes the object to be enclosed, and “>” immediately follows it. Synonymous with <less-than-sign> and <greater-than-sign>. [POSIX.2]

ANSI. American National Standards Institute.

API. Application program interface.

APPC. Advanced Program-to-Program Communication.

APPC/MVS. Advanced Program-to-Program Communication/Multiple Virtual Storage.

append. The action that causes data to be added to the end of existing data. [OSF]

application. (1) The use to which an information processing system is put—for example, a payroll application, an airline reservation application, a network application. (2) The use of capabilities (services or facilities) provided by an information system specific to the satisfaction of a set of user requirements. [POSIX.0] (3) A collection of software components that users buy and install to perform particular types of work on a computer. (4) The combination of an application program and its associated data. (5) See also *application program*.

application development system. A system used to develop user-written application programs that are used later in a production system.

application program. (1) A program written for or by a user that applies to a particular application. (2) In data communication, a program used to connect and communicate with terminals in a network, enabling users to perform application-oriented activities. (3) A program used to perform an application or part of an application. [OSF]

application program interface (API). (1) The formally defined program language interface between an IBM system control program or licensed program and its user. (2) A set of runtime routines or system calls that allows an application program to use a particular service provided by either the operating system or another application program. [OSF] (3) The interface between the application software and the application platform, across which all services are provided. The application program interface is primarily in support of application portability, but system and application interoperability are also support a communication API. [POSIX.0] (4) Synonymous with *application programming interface*.

application programming interface (API). Synonym for *application program interface (API)*.

apply. In journaling, to place after-images of records into a physical file member.

appropriate privileges. An implementation-defined means of associating privileges with a process with regard to most of the function calls and function call options defined in z/OS UNIX that need special privileges. In the z/OS UNIX implementation, appropriate privilege is defined as superuser authority. A trusted or privileged attribute is an attribute associated with a started procedure address space and with any process associated with the address space.

archive. (1) To store programs and data for safekeeping. [OSF] (2) A copy of one or more files or

a copy of a database that is saved in case the original data is damaged or lost. (3) Synonymous with backup, backup copy.

argument. (1) An independent variable. [I][A] (2) Any value of an independent variable, for example, a search key; a number identifying the location of an item in a table. [I][A] (3) A parameter passed between a calling and a called program. (4) A parameter passed to a utility as the equivalent of a single string in the *argv* array created by one of the POSIX.1 **exec** functions. An argument is one of the options, option-arguments, or operands following the command name. [POSIX.2] (5) Numbers, letters, or words that expand or change the way a command works. [OSF] (6) See *command-line argument*.

argument list. A string of arguments.

arithmetic expression. (1) An expression that contains arithmetic operations and operands that can be reduced to a single numeric value. [T] Synonymous with arithmetic statement. (2) A sequence of operations yielding a value. For example, an arithmetic operation such as $a + b * c$ is an arithmetic expression because the result of the operation is a single value. (3) One or more arithmetic operators and arithmetic primaries, the evaluation of which produces a numeric value. An arithmetic expression can be an unsigned arithmetic constant, the name of an arithmetic constant, a reference to an arithmetic variable, array element, or function, or a combination of such primaries formed by using arithmetic operators and parentheses. [OSF] See also *arithmetic operator*. (4) In assembler, a conditional assembly expression that is a combination of arithmetic terms, arithmetic operators, and paired parentheses.

arithmetic operator. A symbol that directs a compiler to perform an arithmetic operation. [OSF]

array. (1) A number of items stored together, which a user can quickly retrieve by supplying the correct index. Both **awk** and the z/OS shells support arrays. (2) A variable that contains an ordered group of data objects. All objects in an array have the same data type. [OSF] (3) An arrangement of data in one or more dimensions, such as a list, table, or multidimensional arrangement of items. (4) In programming languages, an aggregate that consists of data objects, with identical attributes, each of which may be uniquely referenced by subscripting. [I] (5) See also *variable*.

ASCII. ASCII, which stands for American National Standard Code for Information Interchange, is a standard code using a coded character set consisting of 7-bit coded characters (8 bits including parity check), that is used for information interchange among data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters.

assembler. (1) A computer program that converts assembler language instructions into object code. (2) A computer program for (a) translating programs expressed in an assembly language into a machine language and (b) linking subroutines. (3) Synonymous with assembler program. (4) See also *assembler language*.

assembler language. (1) A source language that includes symbolic machine language statements in which there is a one-to-one correspondence with the instruction formats and data formats of the computer. (2) A symbolic programming language in which the set of instructions includes the instructions of the machine and whose data structures correspond directly to the storage and registers of the machine. [OSF] (3) See also *assembler*.

associativity. The order for grouping operands with an operator (either left to right or right to left). [OSF]

asterisk. The character “*”. [POSIX.2]

asynchronous. (1) Without regular time relationship; unexpected or unpredictable with respect to the execution of a program’s instructions. (2) Pertaining to transactions that are processed independently of and simultaneously with the continuing work of the requesting user or application. For example, the keyboard of a terminal user who requests an asynchronous transaction remains unlocked while the transaction is being processed. (3) Contrast with *synchronous*.

asynchronous device. A device using asynchronous data transmission.

asynchronous transmission. Data transmission for a character or block of characters, in which the bits are sent one after another, with a start bit and a stop bit that mark the beginning and end of a data unit. The interval between data units can be uneven—another character might follow immediately or there might be an idle interval before another character is sent. Contrast with *synchronous transmission*.

attach. To create a task that can be executed asynchronously with the execution of the mainline code.

attribute. (1) A named property of an entity. (2) A characteristic or property of one or more objects. For example, the attribute for a displayed field could be *blinking*. [OSF] (3) A property or characteristic of one or more entities—for example, color, weight, sex. [T]

authorize. To grant to a user the right to communicate with or make use of a computer system or display station. [OSF]

automatic conversion. For Enhanced ASCII functionality, the automatic conversion of text data from EBCDIC to ASCII, or from ASCII to EBCDIC as part of using internationalized applications developed on (or

for) ASCII platforms and ported to z/OS platforms. See also *file tag* and *program CCSID*.

automatic variable. A variable allocated on entry to a routine and deallocated on the return.

automount rule. A generic or specific entry in an automount map file.

auxiliary storage. (1) In a data processing system, a storage device that is not central storage. [I][A] (2) Storage that supplements other storage. (3) Data storage other than central storage—for example, storage on magnetic tape or direct access devices. (4) Synonymous with external storage, secondary storage. (5) Contrast with *central storage*.

availability. (1) The ratio of the total time a functional unit is capable of being used to the total time the functional unit is required for use. [A] (2) In computer security, the property of being accessible and usable on demand by an authorized subject. (3) The ability of a functional unit to be in a state to perform a required function under given conditions at a given instant of time or over a given time interval, assuming that the required external resources are provided. (4) The degree to which a system or resource is ready when needed.

awk. A file processing language that is well suited to data manipulation and retrieval of information from text files. **awk** is named for its creators, Alfred V. Aho, Peter J. Weinberger, and Brian W. Kernighan.

B

background. (1) In multiprogramming, the conditions under which low-priority programs are executed. (2) A mode of program execution in which the shell does not wait for program completion before prompting the user for another command. [OSF] Synonym for *background job*. (3) Contrast with *foreground*.

background job. An MVS job that the MVS initiator selects from a queue and starts running. Synonymous with background, batch job.

background process. (1) A process that is a member of a background process group. [POSIX.1] (2) A process that does not require operator intervention but can be run by the computer while the workstation is used to do other work. (3) A process that an interactive user starts running and that cannot interact with the user. An interactive user can move a background process to the foreground. (4) See also *background process group*, *background processing*.

background process group. Any process group, other than a foreground process group, that is a member of a session that has established a connection with a controlling terminal. [POSIX.1]

background processing. (1) The execution of lower-priority computer programs when higher-priority programs are not using the system resources. [I][A] (2) See also *background process*, *background process group*. (3) Contrast with *foreground processing*.

backslash. The character “\”—also known as a *reverse solidus*. [POSIX.2] The backslash enables a user to escape the special meaning of a character. That is, typing a backslash before a character tells the system to ignore any special meaning the character might have.

backspace. A character that normally causes printing (or displaying) to occur one column position previous to the position about to be printed. The <backspace> is the character designated by “\b” in the C language binding. It is unspecified whether this character is the exact sequence transmitted to an output device by the system to accomplish the backspace function. The <backspace> character defined here is not necessarily the ERASE special character defined in POSIX.1. [POSIX.2]

backup. Pertaining to a system, device, file, or facility that can be used in the event of a malfunction or loss of data.

base ACL entry. Same as permission bits (owner, group, other). The permissions can be changed using *chmod*. They are not physically part of the ACL.

base control program (BCP). A mainframe operating system, such as z/OS.

basename. (1) The final, or only, filename in a pathname. [POSIX.2] The basename is the part of a pathname that remains after all directory names are removed. For example, for the pathname **dir1/dir2/file.c**, the basename is **file.c**. (2) The last element to the right of a full pathname. (3) A filename specified without its parent directories.

base register. A general-purpose register that the programmer chooses to contain a base address. Synonymous with base address register.

basic mode. A central processor mode that does not use logical partitioning. Contrast with *logically partitioned (LPAR) mode*.

basic regular expression. A pattern (sequence of characters or symbols) constructed according to the rules defined in POSIX.2 2.8.3. [POSIX.2]

batch. (1) An accumulation of data to be processed. (2) A group of records or data processing jobs brought together for processing or transmission. (3) Pertaining to activity involving little or no user action. (4) Contrast with *conversational*. (5) Synonym for *batch job*.

batch job. (1) An MVS job that the MVS initiator selects from a queue and starts running. (2) A job that is

grouped with other jobs as input to a computing system. (3) Synonymous with background job, batch, batched job.

batch mode. The condition established so that batch processing can be accomplished.

batch processing. (1) The processing of data or the accomplishment of jobs accumulated in advance in such a manner that the user cannot further influence processing while it is in progress. [I][A] (2) The processing of data accumulated over a period of time. [A] (3) The technique of executing a set of computer programs such that each is completed before the next program of the set is started. [A] (4) In realtime systems, the processing of related transactions that have been grouped together. (5) Loosely, the execution of computer programs serially. [A] (6) The sequential input of computer programs or data. (7) A processing method in which a program or programs process records with little or no operator action in a background process. [OSF] (8) Contrast with *interactive processing*. (9) See also *background processing*.

baud. (1) A unit of signaling speed equal to the number of discrete conditions or signal events per second. For example, 1 baud equals one-half dot cycle per second in Morse code, 1 bit per second in a train of binary signals, and one 3-bit value per second in a train of signals that can each assume one of eight different states. [A] (2) In asynchronous transmission, the unit of modulation rate corresponding to one unit interval per second; that is, if the duration of the unit interval is 20 milliseconds, the modulation rate is 50 baud. [A] (3) The number of changes in signal levels, frequency or phase per second on a communication channel. If each represents 1 bit of data, baud is the same as bits per second. However, it is possible for one signal change (1 baud) to equal more than 1 bit of data. [OSF]

binary. (1) Pertaining to a selection, choice, or condition that has two possible different values or states. [I][A] (2) Pertaining to a fixed radix numeration system having a radix of two. [I][A] (3) Pertaining to a system of numbers to the base 2; the binary digits are 0 and 1. [I][A] (4) Pertaining to a choice of two conditions, such as on-off or yes-no. [OSF]

binary data. (1) Any data not intended for direct human reading. Binary data may contain "unprintable" characters, outside the range of text characters. (2) A type of data consisting of numeric values stored in bit patterns of 0s and 1s. Binary data can cause a large number to be placed in a smaller space of storage.

binary file. A file that contains codes that are not part of the character set. Binary files utilize all 256 possible values for each byte in the file. See also *text file*.

binder. Synonym for *linkage editor*.

blank. One of the characters that belong to the *blank* character class as defined via the LC_CTYPE category

in the current locale. In the POSIX locale, a <blank> is either a <tab> or a <space>. [POSIX.2]

block. (1) A string of records, a string of words, or a character string formed for technical or logical reasons to be treated as an entity. (2) A collection of contiguous records recorded as a unit. Blocks are separated by interblock gaps, and each block may contain one or more records. [A] (3) A group of contiguous records or data that is recorded or processed as a unit. [OSF] Synonymous with physical record. (4) A collection of contiguous records, recorded as a unit and transferred between central storage and auxiliary storage. (5) In data communication, a group of records that is recorded, processed, or sent as a unit. [OSF] (6) In programming languages, a compound statement that coincides with the scope of at least one of the declarations contained within it. A block may also specify storage allocation or segment programs for other purposes. [I]

block special file. (1) A special file (for a block device) that provides access to an input or output device and is capable of supporting a file system. (2) A file that refers to a device. A block special file is normally distinguished from a character special file by providing access to the device in a manner such that the hardware characteristics of the device are not visible. [POSIX.1] (3) A file listing the usage of blocks on a disk. [OSF] (4) Synonymous with block file. (5) See also *character special file*, *special file*.

Boolean. (1) Pertaining to the processes used in the algebra formulated by George Boole. [A] (2) A value of 0 or 1 represented internally in binary notation.

boot. (1) The initialization procedure that causes an operating system to begin operation. (2) To prepare a computer system for operation by loading an operating system. (3) Synonymous with initial program load (IPL).

Bourne Shell. The shell based on UNIX V. See also *shell*.

braces. The characters "{" (*left brace*) and "}" (*right brace*), also known as *curly braces*. When used in the phrase "enclosed in (curly) braces," the symbol "{" immediately precedes the object to be enclosed, and "}" immediately follows it. Synonymous with <left-brace> and <right-brace>. [POSIX.2] Synonymous with curly brackets.

brackets. The characters "[" (*left bracket*) and "]" (*right bracket*), also known as *square brackets*. When used in the phrase "enclosed in (square) brackets," the symbol "[" immediately precedes the object to be enclosed, and "]" immediately follows it. Synonymous with <left-square-bracket> and <right-square-bracket>. [POSIX.2]

breakpoint. A place in a computer program, usually specified by an instruction, where execution may be interrupted by external intervention or by a monitor program.

break condition. In the TTY subsystem, a character framing error in which the data is all zeros. [OSF]

break statement. A C language control statement that contains the word **break** and a semicolon.

browse. (1) To look at the records in a file. (2) To look at information without changing it. See also *scan*. (3) To read records in sequential order.

buffer. (1) An area of storage, temporarily reserved for performing input or output, into which data is read or from which data is written. [OSF] (2) An area used to store data read in an input operation or written in an output operation. (3) An area of storage used to hold text. The contents of a file a user is editing can be stored in a "text buffer"; the commands the user runs while using the editor change the contents of the buffer. Typically, the system writes the contents of the buffer back to the original file when the user is finished editing. (4) A routine or storage used to compensate for a difference in rate of flow of data, or time of occurrence of events, when transferring data from one device to another. [A] (5) To allocate and schedule the use of buffers. [A]

build. To convert a product from source code to a binary or executable software product.

built-in shell command. Certain shell commands are built into the shell in order to improve performance of shell scripts or to access the shell's internal data structures and variables. These internal commands have semantics that cannot be distinguished from external commands. A subset of these commands includes special built-in commands. Syntax errors in special built-in commands may cause a shell executing that command to terminate, while syntax errors in regular built-in commands do not cause the shell executing that command to terminate.

byte. (1) Eight bits of data storage. [OSF] (2) The unit of data storage large enough to hold any member of the basic character set of the execution environment. It is possible to express the address of each individual byte of an object uniquely. A byte is composed of a contiguous sequence of bits, the number of which is implementation defined. The least significant bit is called the *low-order* bit; the most significant is called the *high-order* bit. [POSIX.1] (3) A string consisting of a certain number of bits, usually eight, treated as a unit, and representing a character. (4) A group of eight adjacent binary digits representing one EBCDIC character. (5) A sequence of eight adjacent binary digits that are used as a unit.

C

c. (1) In X.25 communication, data optionally included in the clear-request packet by the user application. [OSF] (2) See *C language*.

call. (1) The action of bringing a computer program, a routine, or a subroutine into effect, usually by specifying the entry conditions and jumping to an entry point. [I][A] (2) To activate a program or procedure, usually by specifying the entry conditions and jumping to an entry point. [OSF] Contrast with *load*. (3) In data communication, the action necessary in making a connection between two stations on a switched line. (4) In communication, a conversation between two users.

callable service. (1) A request by an active process for a service by the system kernel. (2) A call to receive z/OS UNIX and other BCP services. (3) Synonymous with *syscall*, *system call*.

cancel. (1) To end a task before it is completed. (2) An action that removes the current panel without processing it and returns the dialog to the previous panel in the hierarchy.

canonical mode. A *tty* input processing mode where input is collected and processed one line at a time. [OSF] There are three CLISTs associated with canonical mode drivers: store output to a terminal, store raw input, store cooked data. I/O processing is asymmetric. Synonymous with *cooked mode*, *line mode*. Contrast with *noncanonical mode*.

carriage control character (CC character). The first character of an output record (line) to be printed; it determines how many lines should be skipped before that line is printed.

carriage return (CR). (1) A keystroke generally indicating the end of a command line. (2) In text data, the action that indicates to continue printing at the left margin of the next line. [OSF] (3) A character that in the output stream causes printing to start at the beginning of the same physical line in which the <carriage return> occurred. The <carriage return> is the character designated by the "r" in the C language binding. It is unspecified whether this character is the exact sequence transmitted to an output device by the system to accomplish the movement to the beginning of the line. [POSIX.2]

Cartesian product. The product of two sets A and B (expressed as "AxB"), in which x is an element of A and y is an element of B, and x and y are themselves part of a set of ordered pairs (x, y). For example, if A={1,2} and B={3,4}, then AxB={(1,3), (1,4), (2,3), (2,4)}.

case-sensitive. Pertaining to the ability to distinguish between uppercase and lowercase letters.

catalog. (1) A directory of files and libraries, with reference to their locations. [I][A] (2) To enter information about a file or a library into a catalog. [I][A] (3) The collection of all data set indexes that are used by the control program to locate a volume containing a specific data set. (4) To enter information about a file or a library into a catalog. [I][A]

CBPDO. (Custom-built Product Delivery Option) a software delivery package consisting of uninstalled products and unintegrated service. Installing the individual z/OS elements and features requires the use of SMP/E. CBPDO is one of the two entitled methods for installing z/OS, ServerPac being the other.

CCSID. See *coded character set identifier*.

CDS (couple data set). See *Couple Data Set (CDS)*

CECP. Country-extended code page.

central processor (CP). The part of the computer that contains the sequencing and processing facilities for instruction execution, initial program load, and other machine operations.

central storage. Storage that is an integral part of the processor unit. Central storage includes both main storage and the hardware system area. UNIX-experienced users refer to central storage as *memory*.

character. (1) A letter, digit, or other symbol. (2) A letter, digit, or other symbol that is used as part of the organization, control, or representation of data. A character is often in the form of a spatial arrangement of adjacent or connected strokes. [A] (3) A sequence of one or more bytes representing a single graphic symbol. The term corresponds in the C Standard to the term *multibyte character*, noting that a single-byte character is a special case of a multibyte character. In POSIX, however, a character has no necessary relationship with storage space, and *byte* is used when storage space is discussed. [POSIX.1] (4) A member of a set of elements that is used for the representation, organization, or control of data. Characters may be letters, digits, punctuation marks, or other symbols. [T]

character class. (1) A named set of characters sharing an attribute associated with the name of the class. The classes and the characters that they contain are dependent on the value of the LC_CTYPE category in the current locale. [POSIX.2] (2) Ranges of characters that match a single character in the input stream. [OSF] (3) A set of characters enclosed in sequence, or square, brackets. [OSF]

character constant. (1) A constant with a character value. (2) In C language, a character or an escape sequence enclosed in single quotation marks. Some compilers allow more than one character or escape sequence in a character constant. [OSF]

character conversion table. A table that converts one or more characters to alternative characters using hexadecimal encoding for the character sets. The character sets are defined in code pages.

character mode. Synonym for *noncanonical mode*.

character set. (1) A defined collection of characters. (2) All the valid characters for a programming language or for a computer system. (3) A group of characters used for a specific reason—for example, the set of characters a printer can print or a keyboard can support.

character special file. (1) A special file that provides access to an input or output device. The character interface is used for devices that do not use block I/O. (2) A file that refers to a device. One specific type of character special file is a terminal device file. Other character special files have no structure defined by POSIX.1, and their use is unspecified by POSIX.1. [POSIX.1] The only character special file supported by the z/OS UNIX implementation is the pseudo-TTY, or pseudoterminal. (3) See also *block special file*, *special file*.

character string. A sequence of consecutive characters.

character type. A data type that consists of alphanumeric characters. See also *data type*.

checksum. The sum of a group of data associated with the group and used for checking purposes. [T]

child. See *child device*, *child process*, *child resource*, *child segment*.

child device. A hierarchical location term that indicates what can be connected to a parent device. For example, an SCSI disk can be a child device of an SCSI adapter.

child process. A process created as a result of a fork. The child process receives a copy of the parent's storage and inherits open files. Execution in the child continues at the instruction following the fork. Contrast with *parent process*. See also *fork*, *process*.

child resource. A secured resource, either a file or library, that uses the user list of a parent resource. A child resource can have only one parent resource.

child segment. In a hierarchical database, a segment immediately below its parent segment. A child segment has only one parent segment. See also *parent segment*.

choice. (1) An item in a selection field that users may select. A choice can be resented as text, a symbol (number or letter), or an icon (a pictorial symbol). (2) An option in a popup or menu used to influence the operation of the system.

CICS. Customer Information Control System.

circumflex. The character “^”. [POSIX.2]

C language. A language used to develop software applications in compact, efficient code that can be run on different types of computers with minimal change.

class. (1) Any category to which things are arranged or defined. (2) A regular expression that matches any one of a set of characters. (3) The I/O characteristics of a device.

C library. A system library that contains common C language subroutines for file access, string operators, character operations, memory allocation, and other functions.

client. (1) A user. (2) A consumer of resources or services. [OSF] (3) A functional unit that receives shared services from a server. [T] (4) A system that is dependent on a server to provide it with programs or access to programs. (5) Contrast with *server*.

client process. In the client-server model of communication, a process that requests services from a server process. [OSF] Contrast with *server process*.

CLIST. Command list used in TSO/E.

close. A data manipulation function that ends the connection between a file and a program. Contrast with *open*.

code. (1) Instructions written for a computer. (2) A set of items, such as abbreviations, representing the members of another set. [A] (3) Loosely, one or more computer programs, or part of a computer program. (4) A representation of a condition, such as an error code. (5) A set of unambiguous rules specifying the manner in which data can be represented in a discrete form. [OSF] Synonymous with coding scheme. (6) To write instructions for the computer. (7) To write a routine. [A] (8) To represent data or a computer program in a symbolic form that can be accepted by a data processor. [I][A]

coded character set identifier (CCSID). For Enhanced ASCII functionality, a 16-bit value is a number that represents a character set used by file tagging. It identifies the current character set of text strings within a program. This is stored in the file tag of new files or used for the automatic conversion of old files when autoconversion is in effect. See also *automatic conversion* and *file tag*.

code page. (1) A table showing codes assigned to character sets. (2) An assignment of graphic characters and control function meanings to all code points. (3) Arrays of code points representing characters that establish ordinal sequence (numeric order) of characters. [OSF] (4) A particular assignment of

hexadecimal identifiers to graphic elements. (5) Synonymous with code set. (6) See also *code point*, *extended character*.

code point. A 1-byte code representing one of 256 potential characters.

collating element. The smallest entity used to determine the logical ordering of strings. A collating element shall consist of either a single character, or two or more characters collating as a single entity. The value of the LC_COLLATE category in the current locale determines the current set of collating elements. See also *collation sequence*. [POSIX.2]

collating sequence. Synonym for *collation sequence*.

collation sequence. (1) A specified arrangement used in sequencing. [I][A] (2) The sequence in which characters are ordered within the computer for sorting, combining or comparing. (3) The relative order of collating elements as determined by the setting of the LC_COLLATE category in the current locale. The character collation sequence defines the relative order of all collating elements such that each element occupies a unique position in the sequence. This is also the default sorting sequence. Multilevel sorting is accomplished by assigning elements one or more collation weights, up to the limit {COLL_WEIGHTS_MAX}. On each level, elements may be given the same weight or can be omitted from the sequence. Strings that collate equally using the first assigned weight (primary ordering) are then compared using the next assigned weight (secondary ordering), and so on. See also *equivalence class*. [POSIX.2] (4) Synonymous with collating sequence.

colony address space. An address space in which a physical file system (PFS) can be initialized. The address space can be viewed as logical extensions to the kernel address space. Some PFSes cannot be initialized in colonies; for example, character special files, pipes, and file systems defined with a type of INET or HFS.

column. The vertical arrangement of characters or other expressions. [A] Contrast with *row*.

command. (1) A directive to the shell to perform a particular task. [POSIX.2] (2) A unit of input to an interactive program. The syntax of a command depends on the program processing it. A frequently used program is the shell, the input of which consists mostly of command lines. (3) A request to perform an operation or run a program. When parameters, arguments, flags, or other operands are associated with a command, the resulting character string is a single command. [OSF] (4) A request from a terminal for the performance of an operation or execution of a program. (5) A character string from a source external to a system that represents a request for system action. (6) One or more words that tell the system to perform an operation or

run a program. It consists of a name and, optionally, one or more operands. (7) The typed name and parameters associated with an action that can be performed by an application. A command is one form of action request. Users type in the command and enter it. The computer performs the action requested by the command name. (8) An instruction a user can type on the command line or in a CLIST. (9) A common action users request to interact with the command area.

command alias. An abbreviation of a long command line or a new name for a command. [OSF]

command history. A feature that stores commands and allows you to edit and reuse them. [OSF]

command interpreter. A program that reads the commands that you type in and then executes them. When you are typing commands into the computer, you are actually typing input to the command interpreter. The interpreter then decides how to perform the commands that you have typed. The shell is an example of a command interpreter. Synonymous with command language interpreter.

command language interpreter. Synonym for *shell*.

command line. (1) The area of the screen where commands are displayed as they are typed. [OSF] (2) The position after the arrow (==>) at the top or bottom of the panel where the user can type a command. (3) A unit of input to the shell, consisting of a program name and possibly command line arguments, separated by white space and terminated by a newline.

command-line argument. A part of a command line, delimited by white space. Arguments are used to specify detailed behavior to a program. They are usually either command line options selecting variations in program operation, or pathnames of files to be processed.

command list (CLIST). (1) A data set in which commands and possibly subcommands and data are stored for subsequent execution. (2) A sequential list of commands, control statements, or both that is assigned a name and executed when that name is invoked.

command mode. (1) A state of a system or device in which the user can enter commands. (2) In an editing session, the mode wherein the editor is waiting for the user to enter a command.

command name. (1) The first term in a command. It requests a specific action, and it is usually followed by operands. (2) The first or principal term in a command. A command name does not include parameters, arguments, flags, or other operands. [OSF]

command processor. A program executed to perform an operation specified by a command.

command substitution. The ability to capture the output of any command as an argument to another

command by placing that command line within grave accents (`). The shell first runs the command or commands enclosed within the grave accents and then replaces the whole expression, including grave accents, with their output. This feature is often used in assignment statements.

comment. (1) A statement used to document a program or file. Comments include information that may be helpful in running a job or reviewing an output listing. (2) In programming languages, a language construct for the inclusion of text in a program that has no impact on the execution of the program. (3) An English language description of what a program does. In the shell and in **awk**, a comment is introduced with a # character and ends at the end of the line.

commit. To make permanent all changes that have been made to the database file since the last commitment operation and to unlock the records so they are available to other users. [OSF] Contrast with *reject*.

Common INET (CINET) physical file system. A physical file system layer for the address family AF_INET that multiplexes sockets across several other physical file systems, or transports.

common link access to workstations (CLAW). A protocol supporting a RISC System/6000[®] attached by channel to an AIX/ESA[®] system. CLAW is also used to connect z/OS to a RISC System/6000. The CLAW protocol improves host system performance by reducing the number of I/O interrupts to the host.

communication. The process of sending or receiving data between two points of a network.

compatibility. (1) The ability to perform tasks identically in different environments without major modifications. [OSF] (2) The capability of a functional unit to meet the requirements of a specified interface. [OSF]

compatibility mode. A mode of processing in which the IEAIPsxx and IEAICSxx parmlib members determine system resource management. See also *goal mode*.

compilation unit. A portion of a computer program sufficiently complete to be compiled correctly. [A]

compile. (1) To convert a computer program expressed in a problem-oriented language into a computer-oriented language. (2) To prepare a machine language program from a computer program written in another programming language by making use of the overall logic structure of the program, or generating more than one computer instruction for each symbolic statement, or both, as well as performing the function of an assembler. [A] (3) To convert a program written in a high-level programming language into an intermediate language, an assembler language, or a machine

language. (4) The computer actions required to transform a source file into an executable object file. [OSF]

compiler. (1) A program that translates a source program into an executable program (an object program). (2) A program that translates a computer program expressed in a problem-oriented language into a machine language program to be executed at a later time. Synonymous with compiling program.

compiler options. Keywords that can be specified to control certain aspects of compilation. Compiler options can control the nature of the load module generated by the compiler, the types of printed output to be produced, the efficient use of the compiler, and the destination of error messages.

component. (1) One part of a structured type or value, such as an array element or a record field. [OSF] (2) Hardware or software that is part of a functional unit. (3) The functional part of an operating system; for example, the scheduler or supervisor.

concatenate. (1) To link together. (2) To join two character strings.

concurrent. (1) Pertaining to the occurrence of two or more activities within a given interval of time. (2) Contrast with *consecutive*, *sequential*.

condition. An expression in a program or procedure that can be evaluated to a value of either true or false when the program or procedure is running.

conditional. An instruction that tests for certain circumstances in order to run particular commands. [OSF]

condition code. Synonym for *flag*.

condition variable. In DCE, a synchronization object used in conjunction with a mutex for explicit communication among threads. A condition variable allows a thread to block until some event happens. [OSF]

configuration. (1) The group of machines, devices, and programs that make up a data processing system or network. (2) The collection and arrangement of programs and devices that make up a particular data processing system or subsystem. (3) The act of making a subsystem, or a set of subsystems, available for use by a running operating system. (4) The set of configured subsystems in an operating system.

configure. To describe to a system the devices and optional features installed on a system.

conformance document. A document provided by an implementer (such as IBM) that contains implementation details as described in the POSIX.1 standard 1.3.1.2. [POSIX.1]

connection. (1) In data communication, an association established between functional units for conveying information. [I][A] (2) In response to a request from an application program, the linking of control blocks in such a way that the program can communicate with a particular I/O resource. The connection process includes establishing and preparing the path between the program and the I/O resource.

consecutive. In a process, pertaining to two events that follow one another without the occurrence of any other event between them. [T] Contrast with *concurrent*, *sequential*.

consistent. Pertaining to a file system, without internal discrepancies. [OSF]

console. (1) The main system display station. [OSF] (2) A device name associated with the main system display station. [OSF]

constant. (1) A data item with a value that does not change during program execution. Contrast with *variable*. (2) A fixed or invariable value or data item.

constant field. (1) A field defined by a display format to contain a value that does not change. (2) A fixed field that cannot be used for input or output data.

context. (1) A stated or implied sense in which a thing has meaning, or a category or scope to which it applies. (2) A process being run. The context includes the information about the process that needs to be saved when the context is switched, so that another process receives control. (3) The address space for the process, the hardware registers, and related kernel data structures. Context consists of:

- User-level context: text, data, user stack
- Register context: GPRs, hardware regs, program counters, PSW, and stack pointer.
- System context: static. process table entry, preregion entries, and process control information
- System context: dynamic. Kernel stack, set of system context layers.

contiguous. Touching or joining at the edge or boundary; adjacent. For example, an unbroken consecutive series of storage locations.

control block. A storage area used by a computer program to hold control information. [I]

control character. (1) A character whose occurrence in a particular context specifies a control function. [T] (2) A character in a data stream that indicates the initiation, modification, or termination of a control operation. A control character can be recorded for use in a subsequent action, have a graphic representation in some circumstances, and appear anywhere in the text data. (3) A character, occurring in a particular context, that initiates, modifies or stops any operation that affects the recording, processing, transmission or interpretation of data (such as carriage return, font

change and end of transmission). [OSF] (4) A nonprinting character that performs formatting functions in a text file. [OSF] (5) Contrast with *carriage control character*.

controlled program. A program that has the RACF trusted attribute. Controlled programs can run in environments where sensitive data such as userid or password is handled. Controlled programs have profiles protecting them in the RACF PROGRAM class.

controlling process. The session leader that established the connection to the controlling terminal. If the terminal subsequently ceases to be a controlling terminal for this session, the session leader shall cease to be the controlling process. [POSIX.1]

controlling terminal. (1) An active terminal at which a user is authorized to enter commands that affect system operation. The controlling terminal for any process normally is the active terminal from which the process group for that process was started. A terminal can have no more than one controlling process group and a process group can have no more than one controlling terminal. The controlling process group receives certain interrupt signals from the controlling terminal. [OSF] (2) A terminal that is associated with a session. Each session may have at most one controlling terminal associated with it, and a controlling terminal is associated with exactly one session. Certain input sequences from the controlling terminal cause signals to be sent to all processes in the process group associated with the controlling terminal. [POSIX.1]

control register (CR). A register used for operating system control of relocation, priority interruption, program event recording, error recovery, and masking operations.

control section (CSECT). That part of a program specified by the programmer to be a relocatable unit, all elements of which are to be loaded into adjoining central storage locations.

control statement. (1) A language statement that changes the normal path of execution. [OSF] (2) In programming languages, a statement that is used to alter the continuous sequential execution of statements; a control statement may be a conditional statement such as IF or an imperative statement such as STOP. [T]

conversational. Pertaining to a program or a system that carries on a dialog with a terminal user, alternately accepting input and then responding to the input quickly enough for the user to maintain a train of thought. Contrast with *batch*. See also *interactive*, *transactional*.

conversion. (1) The process of changing from one form of representation to another—for example, to change from decimal representation to binary representation; or to change from ASCII to EBCDIC. (2) A change in the type of value. For example, in some

programming languages when you add values having different data types, the compiler converts both values to the same form before adding them. (3) In programming languages, the transformation between values that represent the same data item but belong to different data types. Information may be lost due to conversion since accuracy of data representation varies among different data types. [I] (4) The process of changing from one method of data processing to another or from one data processing system to another. (5) Sometimes referred to as translation.

conversion table. (1) A table used to replace one or more characters with alternative characters—for example, to convert characters representing an event to those representing a procedure call, characters of a code set to those of another code set, or characters representing a relocated address to those representing an absolute address. (2) A table that maps virtual addresses with real addresses. (3) A table that specifies the mapping of events or event sequences to procedure names. (4) Synonymous with *translate table*, *translation table*.

cooked mode. Synonymous with *canonical mode*.

copy. (1) To read data from a source, leaving the source data unchanged, and to write the same data elsewhere. [I][A] (2) The action by which the user makes a whole or partial duplicate of an already existing data object. [OSF]

counter. A register or storage location used to accumulate the number of occurrences of an event.

country-extended code page (CECP). An 8-bit code page that has a 93-character set on its nationally standardized code points but is extended to the multilingual character set for the national languages of some European countries.

couple data set (CDS). A data set that is created through the XCF couple data set utility and, depending on its designated type, is shared by some or all of the MVS systems in a sysplex. See also *sysplex couple data set*.

coupling facility. A special logical partition that provides high-speed caching, list processing, and locking functions in a sysplex.

coupling services. In a sysplex, the functions of XCF that transfer data and status between members of a group residing on one or more MVS system in the sysplex.

CR. Control register.

cross-system coupling facility (XCF). XCF is an MVS component that provides functions to support cooperation between authorized programs running within a sysplex.

CS. Communications Server.

CSECT. Control section.

C shell. See *shell*.

current directory. The directory that is active and can be displayed with the **pwd** command. Synonymous with current working directory. [OSF]

current file. The file being edited. If multiple windows are in use, the current file is the file containing the cursor. [OSF]

current line. The line on which the cursor is located. [OSF]

current record. (1) The record pointed to by the current line pointer. (2) The record that is currently available to the program.

current thread. In a multi-thread process, the single thread in the context of which **dbx** gives control to the user. Many **dbx** commands operate in the context of the current thread.

current working directory. (1) The directory a user is working with. (2) Synonym for *current directory*. (3) Synonym for *working directory*.

cursor. (1) A movable spot of light on the screen of a display device, usually indicating where the next character is to be entered, replaced, or deleted. (2) A movable symbol (such as an underline) on a display that indicates to the user where the next typed character will be placed or where the next action will be directed. [OSF] (3) A marker that indicates the current data access location within a file. [OSF] (4) A symbol displayed in a panel and associated with an input device. The cursor indicates where input from the device will be placed.

customization. The process of designing a data processing installation or network to meet a user's needs.

D

daemon. A long-lived process that runs unattended to perform continuous or periodic systemwide functions, such as network control. Some daemons are triggered automatically to perform their task; others operate periodically. An example is the **cron** daemon, which periodically performs the tasks listed in the **crontab** file. The MVS equivalent is a *started task*.

DADSM. Direct access device space management.

DASD. Direct access storage device.

data area. A storage area used by a program to hold information.

database. (1) A collection of data fundamental to a system. [A] (2) A collection of data fundamental to an enterprise. [A] (3) A set of data sufficient for a given purpose or for one or several given data processing systems. (4) A collection of interrelated or independent data items stored together without unnecessary redundancy, to serve one or more applications. (5) A collection of data that is managed in such a way that it can be shared by a variety of programming applications and yet be protected from the data stability, recovery, and security problems that accompany data sharing.

data control block (DCB). A control block used by access method routines in storing and retrieving data.

data definition. A program statement that describes the features of, specifies relationships of, or establishes the context of, data. [A] A data definition can also provide an initial value. Definitions appear outside a function or at the beginning of a block statement.

data definition name (ddname). (1) The name of a data definition (DD) statement that corresponds to a data control block that contains the same name. (2) The symbolic representation for a name placed in the name field of a data definition (DD) statement.

data integrity. (1) The condition that exists as long as accidental or intentional destruction, alteration, or loss of data does not occur. [T] (2) Preservation of data for its intended use.

data processing. The systematic performance of operations upon data; for example, handling, merging, sorting, computing. [I][A]

data security. The protection of data against unauthorized disclosure, transfer, modification, or destruction, whether accidental or intentional. [CCITT/ITU]

data set. (1) The major unit of data storage and retrieval in the operating system, consisting of a collection of data in one of several prescribed arrangements and described by control information to which the system has access. (2) The major unit of data storage and retrieval in an MVS system. A hierarchical file system is stored in an HFS data set.

data space. MVS shared memory, parallel to shared memory regions in POSIX. A data space contains data only, which can be shared by multiple address spaces (users) without inadvertently being modified.

data stream. (1) All data transmitted through a channel in a single read or write operation. (2) A continuous stream of data elements being transmitted, or intended for transmission, in character or binary-digit form, using a defined format. (3) All information (data and control information) transmitted over a data channel in a single read or write operation. (4) A continuous stream of data elements being transmitted, or intended

for transmission, in character or binary-digit form using a defined format. Synonymous with stream.

data structure. The syntactic structure of symbolic expressions and their storage characteristics. [T]

data type. (1) In programming languages, a set of values together with a set of permitted operations. [I] (2) The mathematical properties and internal representation of data and functions. The four basic types are integer, real, complex, and logical. (3) An attribute used for defining data as numeric or character. (4) The structural characteristics, features and properties of data that may be directly specified by a programming language—for example, integers and real numbers in FORTRAN, arrays in APL, linked lists in LISP, and character string in SNOBOL. (5) The storage characteristics of a data item.

DBCS. Double-byte character set.

DCB. Data control block

DCE. Distributed Computing Environment.

ddname. Data definition name.

DD statement. Data definition statement.

deadlock. (1) An error condition in which processing cannot continue because each of two elements of the process is waiting for an action by or a response from the other. (2) Unresolved contention for the use of a resource. (3) An impasse that occurs when multiple processes are waiting for the availability of a resource that does not become available because it is being held by another process that is in a similar wait state. (4) A situation that occurs if processes simultaneously attempt to get semaphores (locks) held by the other.

deallocate. (1) To release a resource that is assigned to a specific task. (2) A request to remove the allocation of the specified conversation from the local transaction program. [OSF] (3) Contrast with *allocate*.

debug. To detect, diagnose, and eliminate errors in programs. [T]

debugger. A device used to detect, trace, and eliminate errors in computer programs or software. [OSF]

decimal. Pertaining to a system of numbers to the base 10.

declaration. (1) A description that makes a defined object available to a function or a block. (2) In programming languages, the mechanism for establishing a language object. A declaration normally involves attaching an identifier and allocating attributes to the language object concerned. [I] (3) In a

programming language, a meaningful expression that affects the interpretation of other expressions in that language. [A]

default. (1) A value, attribute or option that is assumed when no alternative is specified by the user. [OSF] (2) An alternative value, attribute, or option that the system supplies when the user does not specify a value.

default ACL. An ACL that is specifically associated with a directory, and which gets inherited by an object created within the directory.

default directory. The directory name supplied by the operating system if none is specified.

default value. A value stored in the system that is used when no other value is specified.

definition. (1) In programming languages, a data type description or a data object description that reserves storage and, sometimes, provides an initial value. [OSF] (2) The means of describing an element of the network.

delete. (1) To remove—for example, to delete a file. (2) To remove an object or a unit of data, such as a character, field, or record. (3) A typing action that deletes the character at the cursor position and shifts the characters that are in the same field and to the right of the cursor one position to the left.

delimiter. A character or sequence of characters that marks the beginning or end of a character string or unit of data. [OSF]

descriptor. A small, unsigned integer that a UNIX system uses to identify an object supported by the kernel. Descriptors can represent files, pipes, sockets, and other I/O streams. They are created, acted on, and deallocated by system calls specific to the object. [OSF]

detach. To stop a thread.

device. (1) A computer peripheral or an object that appears to the application as such. [POSIX.1] (2) A mechanical, electrical, or electronic machine that is designed for a specific purpose and that attaches to your computer, such as a printer, plotter, or disk drive. [OSF] (3) Hardware, such as a terminal, printer, or card reader. (4) Any piece of equipment that can give information to or receive information from your computer. Commonly used devices are hard disk drives, floppy disk drives, video display terminals, and printers.

device driver. A collection of subroutines that control the interface between I/O device controllers and the processor.

device manager. A collection of routines for complex interfaces that acts as an intermediary between drivers and virtual machines; for example, supervisor calls from

a virtual machine are examined by a device manager and routed to the appropriate subordinate device drivers.

device type. The general name for a kind of device sharing the same model number. [OSF]

DFSMS. Data Facility System-Managed Storage.

diagnostic. Pertaining to the detection and isolation of errors in programs and faults in equipment.

digit. (1) A graphic character that represents an integer—for example, one of the characters 0 to 9. (2) A symbol that represents one of the nonnegative integers smaller than the radix. For example, in decimal notation, a digit is one of the characters from 0 through 9. In hexadecimal notation, a digit is one of the characters from 0 through F. (3) Synonymous with numeric character.

direct access storage device (DASD). (1) A storage device in which the access time is independent of the location of the data—for example, a disk or diskette. (2) A storage device that provides direct access to data. [I][A]

direct data set. A data set whose records are in random order on a direct access volume. Each record is stored or retrieved according to its actual address or its address relative to the beginning of the data set. Contrast with *sequential data set*.

directory. (1) A type of file containing the names and controlling information for other files or other directories. (2) A construct for organizing computer files. As files are analogous to folders that hold information, a directory is analogous to a drawer that can hold a number of folders. Directories can also contain subdirectories, which can contain subdirectories of their own. (3) A file that contains directory entries. No two directory entries in the same directory can have the same name. [POSIX.1] (4) A file that points to files and to other directories. (5) An index used by a control program to locate blocks of data that are stored in separate areas of a data set in direct access storage.

directory default ACL. A model ACL that is inherited by subdirectories that are created within the parent directory.

directory entry. An object that associates a filename with a file. Several directory entries can associate names with the same file. Synonymous with link. [POSIX.1]

directory model ACL. See *directory default ACL*.

dirty address space. If an address space requires daemon authority, any program loaded into the address space must be a controlled program. If a program that is not a controlled program is loaded, the address space is marked dirty and cannot perform daemon activities.

Dirty address spaces are also known as dirty environments. See also *controlled programs*.

dirty environment. See *dirty address space*.

disable. (1) To make nonfunctional. In interactive communications, to disconnect or stop a subsystem. Contrast with *enable*. (2) To put a processing unit in the disabled state.

disabled. (1) Pertaining to a state of a processing unit that prevents the occurrence of certain types of interruptions. (2) Pertaining to a logical unit (LU) that has indicated to its system services control point (SSCP) that it is temporarily not ready to establish LU-LU sessions. (3) Pertaining to the state in which a transmission control unit or audio response unit cannot accept incoming calls on a line. (4) Not selectable.

discretionary access control (DAC). A security mechanism that protects information from unauthorized disclosure or modification through owner-controlled access to files.

distributed computing. Computing that involves the cooperation of two or more machines communicating over a network. Data and resources are shared among the individual computers.

Distributed Computing Environment (DCE). A comprehensive, integrated set of services that supports the development, use, and maintenance of distributed applications. The availability of a uniform set of service, anywhere in the network, enables applications to effectively harness the power that tends to lie unused in many networks. DCE is independent of the operating system and network; it provides interoperability and portability across heterogeneous platforms. It provides compatibility with users' existing environments. [OSF]

dollar sign. The character "\$". POSIX permits the substitution of the "currency symbol" graphic defined in ISO/IEC 646 {1} for this symbol when the character set being used has substituted that graphic for the graphic \$. The graphic symbol \$ is always used in the POSIX standard, but not in any monetary sense. [POSIX.2]

domain. (1) In TCP/IP, the naming system used in hierarchical networks. The domain naming system uses the DOMAIN protocol and the named daemon. In a domain system, groups of hosts are administered separately within a tree-structured hierarchy of domains and subdomains. (2) In SNA, the network resources that are under the control of a particular system services control point (SSCP). (3) In SNA, a system services control point (SSCP) and the physical units (PUs), logical units (LUs), links, link stations, and all the associated resources that the SSCP has the ability to control by means of activation requests and deactivation requests. (4) In a network, the resources that are under the control of one or more associated host processors.

dot. (1) A symbol (.) that indicates the current directory in a relative pathname. [OSF] (2) The filename consisting of a single dot character (.). This filename refers to the directory specified by its predecessor. See also *period*. [POSIX.1]

dot-dot. (1) A symbol (..) in a relative pathname that indicates the parent directory. [OSF] (2) The filename consisting solely of two dot characters (..). This filename refers to the parent directory of its predecessor directory. As a special case, in the root directory, dot-dot may refer to the root directory itself. [POSIX.1]

doublebyte character set (DBCS). A set of characters in which each character is represented by two bytes. Languages such as Japanese, Chinese, and Korean, which contain more symbols than can be represented by 256 code points, require doublebyte character sets. Entering, displaying, and printing DBCS characters require special hardware and software support. See also *Hiragana*, *Kanji*, *Katakana*.

double-precision. (1) Pertaining to the use of two computer words to represent a number in accordance with the required precision. [I][A] (2) A specification that causes a floating-point value to be stored internally in the long format.

double-quote. The character "—also known as *quotation mark*. [POSIX.2]

DSECT. Dummy control section.

dub. To make an MVS address space known to UNIX System Services (z/OS UNIX). Once dubbed, an address space is considered to be a "process." Address spaces created by **fork()** are automatically dubbed when they are created; other address spaces become dubbed if they invoke a z/OS UNIX service. Dubbing also applies to MVS tasks. A dubbed task is considered to be a "thread." Tasks created by **pthread_create()** are automatically dubbed threads; other tasks are dubbed if they invoke a z/OS UNIX service. Contrast with *undub*.

dummy control section (DSECT). A control section that an assembler can use to format an area of storage without producing any object code. Synonymous with *dummy section*.

dump. (1) To write at a particular instant the contents of storage, or part of storage, onto another data medium for the purpose of safeguarding or debugging the data. [T] (2) To copy data in a readable format from central or auxiliary storage onto an external medium such as tape or printer. (3) To copy the contents of all or part of virtual storage for the purpose of collecting error information. (4) To copy all or part of the contents of an auxiliary storage device for later restoration of the data to an auxiliary storage device of the same type or another type. (5) The output of such a process. A dump is requested for a specific purpose, such as to allow

other use of the storage, as a safeguard against faults or errors, or in connection with debugging. (6) See also *restore*.

dynamic. Occurring at the time of execution.

dynamic configuration. The addition of a subsystem into a running kernel.

dynamic link library (DLL). A file containing executable code and data bound to a program at load time or run time, rather than during linking. The code and data in a dynamic link library can be shared by several applications simultaneously.

dynamic loading. (1) The loading and resolving of a subsystem module into an executing kernel. (2) The loading of routines into central storage as needed by an executing program. Dynamically loaded routines are not part of the load module of an extended program.

dynamic storage. A device that stores data in a manner that permits the data to move or vary with time such that the specified data is not always available for recovery. [A]

E

EBCDIC. Extended binary-coded decimal interchange code.

EBCDIC character. Any one of the symbols included in the 8-bit EBCDIC set.

echo. (1) In data communication, a reflected signal on a communication channel. (2) In word processing, to print or display each character or line as it is keyed in.

edit. (1) To enter, modify, or delete data. (2) To add, change, delete, rearrange, or modify the form or format of data. [OSF] (3) To check the accuracy of information and to indicate if an error is found. [OSF]

editing. In programming languages, transforming values to the representations specified by a given format. [I]

edit mode. An entry mode in which a user can issue subcommands to enter, modify, delete, or rearrange data.

editor. See *editor program*.

editor program. (1) A program used to enter and modify programs, text, and other types of documents and data. [OSF] (2) A computer program designed to perform such functions as rearrangement, modification, and deletion of data in accordance with prescribed rules. [A] (3) Contrast with *linkage editor*.

effective group ID. (1) The current group ID, but not necessarily the user's own ID. For example, a user logged in under a particular group ID may be able to

change to another group ID. The ID to which the user changes becomes the effective group ID. [OSF] (2) An attribute of a process that is used in determining various permissions, including file access permissions. This value is subject to change during the process lifetime. See also *group ID (GID)*. [POSIX.1]

effective root directory. The point where a system starts when searching for a file. The pathname of the effective root directory begins with a slash (/).

effective user ID. (1) The current user ID, but not necessarily the user's login ID. For example, a user logged in under a login ID may change to another user's ID. The ID to which the user changes becomes the effective user ID until the user switches back to the original login ID. [OSF] (2) An attribute of a process that is used in determining various permissions, including file access permissions. This value is subject to change during the process lifetime. See also *user ID*. [POSIX.1]

element. (1) In a set, an object, entity, or concept having the properties that define a set. [I][A] Synonymous with *member*. (2) The smallest unit of data in a table or array. (3) The component of an array, subrange, enumeration, or set. (4) Any of the bits of a bit string, the octets of an octet string, or the octets by means of which the characters of a character string are represented. [OSF]

empty directory. A directory that contains, at most, directory entries for dot and dot-dot. [POSIX.1]

empty string. A character array whose first element is a null character. Synonymous with *null string*. [POSIX.1]

emulation. (1) The use of programming techniques and special machine features to permit a computing system to execute programs written for another system. (2) Imitation—for example, when one computer imitates the characteristics of another computer.

enable. (1) To make functional. (2) In interactive communication, to load and start a subsystem. (3) To release a processing unit from the disabled state. (4) Contrast with *disable*.

enclave. A transaction that can span multiple dispatchable units (SRBs and tasks) in one or more address spaces and is reported on and managed as a unit.

end of file (EOF). A code that signals that the last record of a file has been read.

end of line. A cursor-movement function that moves the selection cursor to the rightmost position on the current line. For menu panels and entry panels, the end of the line is the rightmost selection field choice or entry field on the line.

end user. (1) A person, device, program, or computer system that utilizes a computer network for the purpose of data processing and information exchange. [T] (2) The ultimate source or destination of information flowing through a system.

enforced lock. A type of lock that a process holds on a region of a file preventing any other process from accessing that region with read or write system calls. In addition, the **create** and **open** commands are prevented from truncating the files. Contrast with *advisory lock*.

enter. (1) To send information to the computer by pressing the Enter key. (2) To type in information on a keyboard and press the Enter key to send the information to the computer. (3) To place on the line a message to be transmitted from a terminal to the computer. (4) A common action that submits panel information to the computer for processing; *enter* tells the computer to perform selected actions on specified objects.

Note: *Select* means to mark a choice on a panel; *enter* sends all designated choices to the computer for processing.

entry. (1) An element of information in a table, list, queue, or other organized structure of data or control information. (2) A single input operation on a workstation. (3) Synonym for *entry point*. (4) In programming languages, a language construct within a procedure, designating the start of the execution sequences of the procedure. [I]

entry point. (1) The address or label of the first instruction executed upon entering a computer program, routine, or subroutine. [I][A] (2) In a routine, any place to which control can be passed. [A]

entry-sequenced data set (ESDS). A data set whose records are loaded without respect to their contents, and whose relative byte addresses cannot change. Records are retrieved and stored by addressed access, and new records are added at the end of the data set.

ENV. The environment variable.

environment. (1) The settings for z/OS shell variables and paths set when the user logs in. These variables can be modified later by the user. [OSF] (2) A block of information passed ("exported") to a command when the command is invoked. This block contains a number of environment variables. The environment provides information that the program may use in its operation, in a form that relieves you of the need to specify it with every command. (3) The set of all factors that may affect how a program behaves. (4) A named collection of logical and physical resources used to support the operation of a function. (5) See also *environment variable*.

environment variable (ENV). (1) A name associated with a string of characters, made available to the programs that you run. Some environment variables used by the z/OS shell are **PATH**, **TMPDIR**, **COLUMNS**, and **LINES**. For example, the **TMPDIR** environment variable holds the name of a directory where shell commands are free to create temporary working files. (2) A variable that describes the operating environment of the process and typically includes information about the home directory, command search path, the terminal in use, and the current time zone (the **HOME**, **PATH**, **TERM**, and **TZ** variables, respectively). (3) A variable included in the current software environment that is available to any called program that requests it.

Epoch. The time 0 hours, 0 minutes, 0 seconds, January 1, 1970, Coordinated Universal Time. See also *seconds since the Epoch*. [POSIX.1]

equivalence class. (1) A grouping of characters or character strings that are considered equal for purposes of collation. For example, many languages place an uppercase character in the same equivalence class as its lowercase form, but some languages distinguish between accented and unaccented character forms for the purpose of collation. (2) A set of collating elements that collate equally on a particular weight level (primary, secondary, or following assigned collation weights). Elements in an equivalence class are typically elements that naturally group together, such as all accented letters based on the same base letter. The order of elements within an equivalence class is determined by the weights assigned on a subsequent level, if any. In regular expressions and pattern matching, only primary equivalence classes are recognized. [POSIX.2]

erase. To remove text from a data medium, leaving the medium available for recording new text. [OSF]

ERE. Extended regular expression.

error. (1) A discrepancy between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition. [I][A] (2) An indicator that is returned by a function when it encounters a system or library error in the process of executing. The object is to allow applications to take an action based on the returned error code. [OSF] (3) See also *error condition*.

error condition. The state that results from an attempt to execute instructions in a computer program that are invalid or that operate on invalid data.

error message. An indication that an error has been detected. [A]

escape. (1) To return to the original level of a user interface. (2) Synonym for *switch*.

escape character (ESC). (1) The control character in a text-control sequence that indicates the beginning of a sequence and the end of any preceding text. (2) In shell

programming and TTY programming, the **** (backslash) character, which indicates that the next character is not intended to have the special meaning normally assigned to it. (3) In general, a character that suppresses or selects a special meaning for one or more characters that follow. [OSF]

escape sequence. (1) A character that is preceded by a **** (backslash) and is interpreted to have a special meaning to the shell. (2) A sequence sent to a terminal to perform actions such as moving the cursor, changing from normal to reverse video, and clearing the screen. The **terminfo** file defines these escape sequences.

ESDS. Entry-sequenced data set.

ESTAE. A type of MVS recovery routine, normally a separate CSECT within a program. Abnormal termination of a program in task mode can be bypassed if an ESTAE macro is coded in the program. ESTAE creates the proper control blocks to inform MVS where to pass execution when the task terminates.

event. (1) The enqueueing or dequeueing of an element. [OSF] (2) An occurrence of significance to a task. [OSF] (3) Information generated either asynchronously from a device or as the side effect of a client request. Events are grouped into types and are not sent to a client by the server unless the client has issued a specific request for information of that type. Events are usually reported relative to a window. [OSF] (4) A synchronization primitive for asynchronous notification of a change of state. (5) An occurrence, or happening, that is significant to a transport user. Events are asynchronous, in that they do not happen as a result of an action taken by the user. [OSF]

exception. (1) An abnormal condition such as an I/O error encountered in processing a data set or a file. (2) One of five types of errors that can occur during a floating-point exception. These are invalid operation, overflow, underflow, division by zero, and inexact results. [OSF] (3) Contrast with *interrupt*, *signal*.

exec. To overlay the current process with another executable program. See also *fork*.

executable. See *executable file*, *executable program*, *executable statement*.

executable file. (1) A file suitable for execution. An executable file may be a program that has been compiled and link-edited, or it may be a shell script. (2) A file that contains programs or commands that perform operations on actions to be taken. (3) A regular file acceptable as a new process image file by the equivalent of the POSIX.1 **exec** family of functions, and thus usable as one form of a utility. The standard utilities described in POSIX.1 as compilers can produce executable files, but other unspecified methods of producing executable files may also be provided. The internal format of an executable file is unspecified, but a

conforming application shall not assume an executable file is a text file. [POSIX.2] (4) A REXX exec. (5) See also *executable program*.

executable module. A module in a partitioned data set (PDS) that can be executed. An executable module that is copied into an HFS file and then given read and execute permissions becomes an executable file. On the other hand, an executable file is not necessarily an executable module.

executable program. (1) A program in a form suitable for execution by a computer. The program can be an application or a shell script. An executable program is equivalent to a z/OS load module. (2) A program that has been link-edited and can therefore be run in a processor. (3) A program that can be executed as a self-contained procedure. It consists of a main program and, optionally, one or more subprograms. (4) See also *executable file*, *load module*.

executable statement. A statement that causes an action to be taken by the program. For example, to calculate, to test conditions, or to alter normal sequential execution. [OSF]

execute. (1) To perform the actions specified by a program or a portion of a program. [T] (2) To perform the actions described in POSIX.2 3.9.1.1. See also *invoke*. [POSIX.2] (3) Synonym for *run*.

execution time. The time during which an operating system is successfully running.

existing file. A file that has been defined and, conceptually, resides on the storage medium. [OSF]

exit. (1) A device driver function that IOS calls. (2) To execute an instruction or statement within a portion of a program in order to terminate the execution of that portion. [T] Such portions of programs include loops, routines, subroutines, and modules. (3) An instruction in a program, routine, or subroutine that causes control to pass to another program, routine, or subroutine. (4) A common action that terminates the current function and returns users to a higher level function. Repeated exit requests return the dialog to the highest level in the panel hierarchy that users are aware of; that is, the point from which all functions provided to the system are accessible. See also *cancel*. (5) Synonym for *exit routine*.

exit routine. A routine that receives control when a specified event occurs, such as an error detected by a system module.

exponent. A number indicating the power to which another number (the base) is to be raised.

exponentiation. The operation in which a value is raised to a power.

export. To make a file system available to a client.

expression. (1) A representation of a value—for example, variables and constants appearing alone or in combination with operators. (2) In programming languages, a language construct for computing a value from one or more operands; for example, literals, identifiers, array references, and function calls. [I] (3) A configuration of signs. [A] (4) A group of constants or variables separated by operators that yields a single value. An expression can be arithmetic, relational, logical, or a character string. (5) An operand or a collection of operators and operands that yields a single result. (6) See *arithmetic expression*.

expression statement. An expression that ends with a ; (semicolon). You can use an expression statement to assign the value of an expression to a variable or to call a function.

extended ACL entry. An ACL entry for individual user or group.

extended binary-coded decimal interchange code (EBCDIC). A coded character set consisting of 8-bit coded characters. [A] See also *American National Standard Code for Information Interchange (ASCII)*.

extended character. A character other than a 7-bit ASCII character. An extended character can be a 1-byte code point with the eighth bit set (ordinal 128–255).

extended regular expression (ERE). A pattern (sequence of characters or symbols) constructed according to the rules defined in POSIX.2 2.8.4. [POSIX.2]

extent. A continuous space on a direct-access storage volume, occupied by or reserved for a particular data set or file.

external. In programming languages, pertaining to a language object that has a scope that extends beyond one module; for example, the entry names of a module. [I]

external link. A special type of symbolic link, a file that contains the name of an object that is outside of the hierarchical file system.

external name. (1) A name that can be referred to by any control section or separately assembled or compiled module; a control section name or an entry name in another module. (2) In a program, a name whose scope is not necessarily confined to one block and its contained blocks.

external variable. A variable accessible to another compilation unit.

extract. To obtain. For example, to extract information from a file.

F

facility. (1) An operational capability, or the means for providing such a capability. [T] (2) A service provided by an operating system for a particular purpose—for example, the checkpoint/restart facility.

field. (1) An area in a record or panel used to contain a particular category of data. [OSF] (2) The smallest identifiable component of a record. (3) In a record, a specified area used for a particular category of data.

field prompt. Information in a panel that identifies a selection field, entry field, or variable information.

FIFO (first-in-first-out). A queuing technique in which the next item to be retrieved is the item that has been in the queue for the longest time. [A] Contrast with *LIFO*.

FIFO special file. (1) A type of file with the property that data written to such a file is read on a first-in-first-out basis. [POSIX.1] (2) A named permanent pipe that allows two unrelated processes to exchange information through a pipe connection. Synonymous with named pipe.

file. (1) A set of related records treated as a unit. (2) A sequence of records. If the file is located in internal storage, it is an internal file; if it is on an input/output device, it is an external file. [OSF] (3) A collection of related data that is stored and retrieved by an assigned name. [OSF] (4) Linear data that can be opened, written, read, and closed. A file can also contain information about the file, such as authorization information. The name used to obtain a file includes the directories in the path to the file. (5) Strings of characters with no additional structure. Structure is assumed only by the processing programs. Files can be located relative to the current directory or by an absolute pathname. (6) An object that can be written to, or read from, or both. A file has certain attributes, including access permissions and type. File types include regular file, character special file, block special file, FIFO special file, and directory. Other types of files may be defined by the implementation. [POSIX.1] In the z/OS UNIX implementation, the file system does not support block special files, but it does support symbolic link files. (7) A collection of information or data that is organized by some method (relative, indexed, or serial, for example) and stored on a device such as a disk. (8) Synonym for *data set*. (9) In word processing, synonym for *document*.

file access permissions. A group of designations that determine who can access a particular file and how the user can access the file. For an HFS file or directory, the permissions are identified in file access permission bits. Appropriate privilege is defined as having superuser authority. No alternate access control mechanisms are provided.

file creation mask. See *mask*.

file default ACL. A model ACL that is inherited by files that are created within the parent directory.

file descriptor. (1) A small unsigned integer that a UNIX system uses to identify a file. A file descriptor is created by a process through issuing an **open** system call for the filename. A file descriptor ceases to exist when it is no longer held by any process. [OSF] (2) A per-process unique, nonnegative integer used to identify an open file for the purpose of file access. [POSIX.1] (3) A small positive number used to identify an open file in I/O operations. By convention, certain file descriptors are used for the same purpose by all programs. (4) See also *standard error (stderr)*, *standard input (stdin)*, *standard output (stdout)*.

file hierarchy. A structure of all the files in the system whereby all the nonterminal nodes are directories and all the terminal nodes are any other type of file. Because multiple directory entries may refer to the same file, the hierarchy is properly described as a “directed graph”.

file lock. A means to limit or deny access to a file by other users. A file lock can be a read lock or a write lock.

file mode. An object containing the file permission bits and other characteristics of a file. [POSIX.1]

file mode bits. A file’s file access permission bits: set-user-ID-on-execution bit (S_ISUID), and set-group-ID-on-execution bit (S_ISGID). [POSIX.2]

file mode creation mask. See *mask*.

| **file model ACL.** See *file default ACL*.

filename. (1) A name assigned or declared for a file. (2) The name used by a program to identify a file. (3) A name consisting of 1 to [NAME_MAX] bytes used to name a file. The characters composing the name may be selected from the set of all character values excluding the slash character and the null character. The filenames dot and dot-dot have special meaning. Synonymous with pathname component. See also *dot*, *dot-dot*. [POSIX.1] (4) See also *label*.

file offset. The byte position in the file where the next I/O operation begins. Each open file description associated with a regular file, block special file, or directory has a file offset. A character special file that does not refer to a terminal device may have a file offset. There is no file offset specified for a pipe or FIFO. [POSIX.1]

file owner. The user who has the highest level of access authority to a file, as defined by the file.

file permission bits. Information about a file that is used, along with other information, to determine if a

process has read, write, or execute/search permission to a file. The bits are divided into three parts: owner, group, and other. Each part is used with the corresponding file class of processes. These bits are contained in the file mode. [POSIX.1]

file pointer. An identifier that indicates a structure containing the filename.

file serial number. A per-file system unique identifier for a file. File serial numbers are unique throughout a file system. [POSIX.1]

file system. (1) A collection of files and directories. (2) The collection of files and file management structures on a physical or logical mass storage device, such as a disk or disk partition. A single device can contain several file systems. (3) A mountable subtree of the directory hierarchy. [OSF] (4) A collection of files and certain of their attributes. A file system provides a name space for file serial numbers referring to those files. [POSIX.1] (5) See also *mountable file system*.

file system owner. The system which coordinates sysplex activity for a particular file system. In many cases, the file system owner is the one system in the sysplex that will have a physical connection to the file system and as such will coordinate I/O for that file system for all users in the sysplex. In addition, the file system owner will coordinate changes to the status of a file system, such as when it is moved or unmounted.

file system type. (1) A design for file management, and the rules for such a design. (2) The name of a program that processes file systems and files.

file tag. A file attribute that identifies the character set of the text data within a file and indicates whether the file is eligible for automatic conversion. See also *automatic conversion* and *program CCSID*.

File Transfer Protocol (FTP). In TCP/IP, an application protocol used for transferring files to and from host computers. FTP requires a user ID and possibly a password to allow access to files on a remote host system. FTP assumes that the Transmission Control Protocol is the underlying protocol.

file tree. The complete directory and file structure of a particular node, starting at the root directory. A file tree contains all local and remote mounts performed on minidisks, directories, and files.

file type. One of the five possible types of files: ordinary file, directory, block device, character device, and first-in-first-out (FIFO or named pipe). See also *file*.

filter. (1) A command that reads standard input data, modifies the data, and sends it to standard output. A pipeline usually has several filters. (2) A program that reads in data (usually text), transforms it in some way, and writes out the result. Examples are a program that

reads in text, converts all letters to uppercase, and then writes out the result; a program that sorts its input and writes out the sorted data; and a program that reads in lines of data, eliminates duplicate lines, and writes out the result. (3) A device or program that separates data, signals, or materials in accordance with specified criteria. [A]

FIPS. Federal Information Processing Standard. For example, FIPS 151-2 lists a group of options and the option choices the system must use.

first-in-first-out (FIFO). A queuing technique in which the next item to be retrieved is the item that has been in the queue for the longest time. [A] Contrast with *last-in-first out (LIFO)*.

fixed-length record. A record having the same length as all other records with which it is logically or physically associated. Contrast with *variable-length record*.

flag. (1) A modifier that appears on a command line with the command name that defines the action of the command. [OSF] (2) An indicator or parameter that shows the setting of a switch. [OSF] (3) A variable indicating that a certain condition holds. [T] (4) A character that signals the occurrence of some condition, such as the end of a word. [A] (5) An internal indicator that describes a condition to the CPU. [OSF] Synonymous with condition code.

floating point. Pertaining to a method of encoding real numbers within the limits of finite precision available on computers. Floating point represents real numbers (that is, values with fractions or decimals) in 32 bits or 64 bits. Floating-point representation is useful to describe very small or very large numbers.

floating-point constant. A constant representation of a floating-point number expressed as an optional sign followed by one or more digits and including a decimal point. [OSF] See also *floating-point number*.

floating-point number. A real number represented by a pair of distinct numerals. The real number is the product of the fractional part, one of the numerals, and a value obtained by raising the implicit floating-point base to a power indicated by the second numeral. [OSF] See also *floating-point constant*.

floating-point register (FPR). A register used to manipulate data in a floating-point representation system. [I][A]

fold. To translate the lowercase characters of a character string into uppercase.

foreground. (1) A process that an interactive user starts running and that can interact with the user. An interactive user can move a foreground process to the background. (2) A mode of program execution in which the shell waits for the program specified on the command line to complete before responding to user

input. (3) In multiprogramming, the environment in which high-priority programs are executed. (4) The interactive execution of programs and services. (5) The environment in which interactive programs are executed. Interactive processors reside in the foreground. (6) Contrast with *background*.

foreground process. (1) A process that is a member of a foreground process group. [POSIX.1] (2) A process that must run to completion before another command is issued to the shell. The foreground process is in the foreground process group.

foreground process group. A process group whose member processes have certain privileges, denied to processes in background process groups, when accessing their controlling terminal. Each session that has established a connection with a controlling terminal has exactly one process group of the session as the foreground process group of that controlling terminal. [POSIX.1] The foreground process group receives the signals generated by the terminal.

foreground process group ID. The process group ID of the foreground process group. [POSIX.1]

foreground processing. (1) The execution of a computer program that preempts the use of computer facilities. [I][A] (2) Contrast with *background processing*.

fork. To create and start a child process. Forking is similar to creating an address space and attaching. Forking creates a copy of the parent process, including open file descriptors.

forked address space. An address space created by a **fork()** function. A forked address space is perceived by MVS to be a batch job.

format. (1) A defined arrangement of such things as characters, fields, and lines, usually used for displays, printouts, or files. [OSF] (2) The pattern that determines how data is recorded. [OSF] (3) To arrange such things as characters, fields, and lines. (4) In programming languages, a language construct that specifies the representation, in character form, of data objects in a file. [I]

formatted file. A file displayed and arranged with particular characteristics, such as line spacing, headings, and number of characters and lines per page. [OSF] Contrast with *unformatted file*.

form-feed. A character that shall cause printing to start on the next page of an output device. The <form-feed> is the character designated by the “\f” in the C language binding. If <form-feed> is not the first character on an output line, the result is unspecified. It is unspecified whether this character is the exact sequence transmitted to an output device by the system to accomplish the movement to the next page. [POSIX.2]

forward. A scrolling action that displays information below that currently visible in a panel body area.

FPR. Floating-point register.

frame. (1) In a high-level data link control (HDLC), the sequence of contiguous bits bracketed by and including opening and closing flag (01111110) sequences. [OSF] (2) A set of consecutive digit time slots in

which the position of each digit time slot can be identified by reference to a frame alignment signal. [OSF]

free storage. Storage that is not allocated. Synonymous with free space.

FRR. Functional recovery routine.

FSCB. Free storage control block.

full pathname. Synonym for *absolute pathname*.

fullword. A word suitable for processing by a given computer, usually treated as a unit. [OSF] Synonymous with computer word, machine word. See also *halfword*.

fully qualified name. A qualified name that includes all names in the hierarchical sequence above the structure member to which the name refers, as well as the name of the member itself. [OSF]

function. (1) A specific purpose of an entity, or its characteristic action. [A] (2) A set of commonly used computations that always returns to its invoker. (3) A data type that contains executable code and returns a single value to the calling function. [OSF] (4) A programming language construct modeled after the mathematical concept of function. A function encapsulates some behavior. It is given some arguments as input, performs some processing, and returns some results. [OSF] (5) A subroutine that returns the value of a single variable and that usually has a single exit, such as subroutines that compute mathematical functions. [T] (6) Synonymous with procedure. (7) See also *procedure*, *subprogram*, *subroutine*.

functional recovery routine (FRR). A recovery routine that is used by the locked programs, the service request blocks, and the supervisor control routines.

function call. An expression that moves the path of execution from the current function to a specified function and evaluates to the return value provided by the called function. A function call contains the name of the function to which control moves and a parenthesized list of arguments.

function key. (1) A key that causes a specified sequence of operations to be performed when it is pressed. Generally used to refer to keys labeled <Fn>, for example, <F1>. (2) A key that requests actions but

does not display or print characters. This definition includes a key that normally produces a printed character, but produces a function instead when used with the code key. [OSF]

G

general-purpose register (GPR). A register, usually explicitly addressable, within a set of registers that can be used for different purposes, for example as an accumulator, as an index register, or as a special handler of data. [I][A] See also *general register*.

general register (GR). A register used for such operations as binary addition, subtraction, multiplication, and division. General registers are used primarily to compute and modify addresses in a program. See also *general-purpose register*.

generation. (1) A means of referencing items in terms of time and ancestry so that an item without antecedents is designated as the first (*n*th) generation and subsequent derivations are designated as *n*–1, *n*–2, and so on. [A] (2) For some remote systems, the translation of configuration into machine language. [A]

GID. Group ID.

global. (1) In programming languages, pertaining to the relationship between a language object and a block in which the language object has a scope extending beyond that block but contained within an encompassing block. [I] (2) Pertaining to information defined in one subdivision of a computer program and used in at least one other subdivision of the program. [OSF] (3) Pertaining to information available to more than one program or subroutine.

goal mode. A mode of processing where the active service policy determines system resource management. See also *compatibility mode*.

GPR. General-purpose register.

GR. General register.

grammar rules. The structure rules in a parser program. See also *parser*.

graphic character. A character that can be displayed or printed.

group. (1) A collection of users who can share access authorities for protected resources. [OSF] (2) A list of names that are known together by a single name. (3) A set of related records that have the same value for a particular field in all records. (4) A series of records logically joined together.

group class. A collection of users who can share access authorities for protected resources. [OSF]

group ID (GID). (1) A unique number assigned to a group of related users. The GID can often be substituted in commands that take a group name as an argument. (2) A nonnegative integer, which can be contained in an object of type *gid_t*, that is used to identify a group of system users. Each system user is a member of at least one group. When the identity of a group is associated with a process, a group ID value is referred to as a real group ID, an effective group ID, one of the (optional) supplementary group IDs, or an (optional) saved set-group-ID. [POSIX.1] (3) Synonymous with group number.

group name. A name that uniquely identifies a group of users to the system.

H

halfword. A contiguous sequence of bits or characters that constitutes half a computer word and can be addressed as a unit. [A] See also *fullword*.

handler. (1) A software routine that controls a program's reaction to specific external events, such as an interrupt handler. (2) A device driver function that the I/O supervisor (IOS) calls.

hardcopy. A printed copy of machine output in a visually readable form such as printed reports, listings, documents, and summaries.

hard link. (1) A mechanism that allows the **In** command to assign more than one name to a file. Both the new name and the file being linked must be in the same file system. [OSF] (2) The relationship between two directory entries that represent the same file; the result of an execution of the **In** utility or the POSIX.1 **link()** function. [POSIX.2]

hardware. (1) Physical equipment as opposed to programs, procedures, rules, and associated documentation. [I][A] (2) Physical equipment used in data processing as opposed to programs, procedures, rules, and associated documentation. [POSIX.0] (3) The equipment, as opposed to the programming, of a system. (4) Contrast with *software*.

header. (1) System-defined control information that precedes user data. (2) The portion of a message that contains control information for the message such as destination fields, originating station, and priority level.

header file. Synonym for *include file*.

heap. A collection of dynamically allocated variables.

help. (1) A common action that provides information about a specific field, an application panel, or information about the help facility. (2) One or more display images that describe how to use application software or how to do a system operation. [OSF]

hexadecimal. (1) Pertaining to a selection, choice, or condition that has sixteen possible different values or states. [I] (2) Pertaining to a fixed-radix numeration system, with radix of sixteen. [I] (3) Pertaining to a numbering system with base of sixteen; valid numbers use the digits 0 through 9 and characters A through F, where A represents 10 and F represents 15.

HFS data set. A hierarchical file system data set, which is used to store, and is essentially identified with, a mountable file system. See also *HFS file*.

HFS file. An object that exists within a mountable file system. Synonymous with POSIX file. See also *HFS data set*.

hierarchical file. See *file*.

high-level language (HLL). (1) A programming language that does not reflect the structure of any particular computer or operating system. [T] (2) A problem-oriented language that requires little knowledge of the computer on which a computer program written in the language is to be run; that facilitates translation of computer programs in this language into several different machine codes; and that usually results in many machine instructions for each statement in the source program; for example, ALGOL, COBOL, COGO, FORTRAN, PL/I, SIMSCRIPT. [T] (3) A language such as COBOL, FORTRAN, APL, and PL/I.

high-order. Most significant; leftmost. [OSF]

Hiperspace™. High performance data space.

Hiragana. A character set of symbols used in one of the two common Japanese phonetic alphabets, the other being Katakana. Each character represents a syllable and itself is represented by a single byte, Hiragana is used for writing authentically Japanese words, as opposed to words of foreign origin. See also *doublebyte character set (DBCS)*, *Kanji*, *Katakana*.

history file. A file in which a record is kept of shell commands that are executed. The default history file is *.sh_history*.

HOME. The environment variable that describes the pathname of a user's home directory.

home directory. (1) The current directory associated with the user at the time of login. [POSIX.2] (2) A directory associated with an individual user. (3) The user's current directory on login or after issuing the *cd* command with no argument.

hook. To configure a dynamic subsystem into the kernel. [OSF]

I

IAR. Instruction address register.

identifier. (1) A name used to refer to a data object. (2) In programming languages, a lexical unit that names a language object, such as the name of an array, record, label, or procedure. [OSF] (3) A sequence of bits or characters that identifies a program, device, or system to another program, device, or system. [OSF]

IEEE. Institute of Electrical and Electronics Engineers.

IF expression. An expression in a procedure that tests for a condition. The action performed by the procedure depends on the result of the test.

if statement. (1) A C language conditional statement that contains the word *if* followed by an expression in parentheses (the condition), a statement (the action), and an optional *else* clause (the alternative action). (2) A conditional statement that specifies a condition to be tested and the action to be taken if the condition is satisfied. [T]

implementation defined. An indication that the implementation defines and documents the requirements for correct program constructs and correct data of a value or behavior. [POSIX.1]

include file. A text file that contains declarations used by a group of functions, programs, or users. Synonymous with header file.

index. (1) A list of the contents of a file or of a document, together with keys or references for locating the contents. [I][A] (2) A table used to locate records in an indexed sequential data set or an indexed file. (3) A table containing the key value and location of each record in an indexed file. [OSF] (4) A computer storage position or register whose contents identify a particular element in a set of elements. [OSF]

indicator. (1) An item of data that can be interrogated to determine whether a particular condition has been satisfied in the execution of a computer program. (2) An internal switch that communicates a condition between parts of a program or procedure. [OSF]

informational message. (1) A message that provides information but does not require a response. (2) A message that is not the result of an error condition.

inherit. To copy resources or attributes from a parent to a child.

inheritance. The act of automatically associating an ACL with a newly created object without requiring administrative action.

initialize. (1) To set counters, switches, addresses, or contents of storage to zero or other starting values at the beginning of, or at prescribed points in, the operation of a computer routine. [A] (2) In programming languages, to set the starting value of a

data object. [OSF] (3) To prepare for use; for example, to initialize a diskette. (4) To prepare the system for operation.

initial program load (IPL). The initialization procedure that causes an operating system to begin operation. Synonymous with *boot*, *load*.

inode. 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. See also *vnode*.

inode number. A number specifying a particular inode file in the file system. Synonymous with *inumber*. See also *inode*.

inoperative. The condition of a resource that has been active, but is not. The resource may have failed, received an INOP request, or is suspended while a reactivate command is being processed.

input. Data to be processed. [OSF]

input device. A physical device that provides data to a computer. [OSF]

input/output (I/O). (1) Pertaining to a device whose parts can perform an input process and an output process at the same time. [I] (2) Pertaining to a functional unit or channel involved in an input process, output process, or both, concurrently or not, and to the data involved in such a process. (3) Pertaining to input, output, or both.

input/output file. A file opened for input and output use. [OSF]

input redirection. The specification of an input source other than standard input/output.

input stream. (1) The sequence of job control statements and data submitted to an operating system on an input unit especially activated for this purpose. (2) The sequence of operation control statements and data given to the system from an input device. [OSF]

inquiry. (1) A request for information in storage. (2) In data communication, a request for information from another system.

installation. (1) The task of adding a device driver to the system and activating the driver so that it can be used. [OSF] (2) In system development, preparing and placing a functional unit in position for use. [T] (3) The process of making the initial product shipment and all updates of a computer system operational. (4) A particular computing system, including the work it does and the people who manage it, operate it, apply it to problems, service it, and use the results it produces.

instruction. A program statement that specifies an operation to be performed by the computer, along with the values or locations of operands. This statement represents the programmer's request to the processor to perform a specific operation. [OSF]

instruction address register (IAR). A system control register containing the address of the next instruction to be executed. [OSF] See also *location counter*.

instructions. Panel body elements that tell users how to interact with a panel and how to continue with the application.

integer. A positive or negative whole number or zero. [OSF]

integer expression. An arithmetic expression with only integer-type values.

integrity. Synonym for *data integrity*.

interactive. (1) Pertaining to an application in which each entry calls forth a response from a system or program. The shell commands can be used interactively. (2) Pertaining to an activity that involves requests and replies, such as between a system user and a program or between two programs. [OSF] (3) Pertaining to a program or system that alternately accepts input and then responds. An interactive system is conversational; that is, a continuous dialog exists between user and system. (4) Pertaining to the exchange of information between a user and a computer. (5) Contrast with *batch*. (6) See also *conversational*, *transactional*.

interactive processing. A processing method in which each system user action causes response from the program or the system. Contrast with *batch processing*.

Interactive System Productivity Facility (ISPF). An IBM licensed program that serves as a full-screen editor and dialog manager. Used for writing application programs, it provides a means of generating standard screen panels and interactive dialogs between the application programmer and terminal user.

interface. (1) A shared boundary. An interface may be a hardware component to link two devices or a portion of storage or registers accessed by two or more computer programs. [A] (2) The shared boundary between two functional units, defined by functional characteristics and other characteristics, as appropriate. [POSIX.0] (3) A shared boundary between two functional units, defined by functional characteristics, common physical interconnection characteristics, signal characteristics, and other characteristics. [OSF] (4) A common boundary, but not of internal connections. An interface may be a hardware component to link two devices, or a portion of storage or registers accessed by two or more computer programs. [OSF] (5) A group of operations that a server can perform. [OSF] (6)

Hardware, software, or both, that link systems, programs, or devices. (7) See also *application program interface (API)*, *shell*.

internationalization. (1) The process of generalizing programs or systems so that they can handle a variety of languages, character sets, and national customs. [OSF] (2) The process of designing and developing a product with a set of features, functions, and options intended to facilitate the adaptation of the product to satisfy a variety of cultural environments. [POSIX.0]

Internet. A wide area network connecting thousands of disparate networks worldwide in industry, education, government, and research. The Internet network is managed by the Network Information Center (NIC) and uses TCP/IP as the standard for transmitting information. Packets sent over Internet must be hardware-independent.

Internet Protocol (IP). In TCP/IP, a protocol that routes data from its source to its destination in an Internet environment. IP provides the interface from the higher level host-to-host protocols to the local network protocols. Addressing at this level is usually from host to host.

interoperability. (1) The capability to communicate, execute programs, or transfer data among various functional units in a way that requires the user to have little or no knowledge of the unique characteristics of those units. [T] (2) The ability of two or more systems to exchange information and to mutually use the information that has been exchanged. [POSIX.0] (3) In SAA usage, the ability to link SAA and non-SAA environments and use the combination for distributed processing.

interprocess communication (IPC). (1) The process by which programs communicate data to each other and synchronize their activities. Semaphores, signals, and internal message queues are common methods of interprocess communication. (2) The exchange of information between processes or threads through semaphores, queues, and shared memory.

interrupt. (1) A suspension of a process, such as execution of a computer program caused by an external event, and performed in such a way that the process can be resumed. [A] (2) A signal sent by an I/O device to the processor when an error has occurred or when assistance is needed to complete I/O. An interrupt usually suspends execution of the currently executing program. [OSF] (3) In data communication, to take an action at a receiving station that causes the sending station to end a transmission. (4) To temporarily stop a process. (5) See also *signal*.

inverse. A square array that results from a mathematical operation on a square array such that the two arrays can be multiplied together to obtain a square array with a determinant of one. [OSF]

invoke. (1) To start a command, procedure, or program. (2) To perform the actions described in POSIX.2 3.9.1.1, except that searching for shell functions and special built-ins is suppressed. See also *execute*. [POSIX.2]

I/O (input/output). (1) Pertaining to a device whose parts can perform an input process and an output process at the same time. [I] (2) Pertaining to a functional unit or channel involved in an input process, output process, or both, concurrently or not, and to the data involved in such a process. (3) Pertaining to input, output, or both.

I/O channel. In a computing system, a functional unit, controlled by a processor, that handles transfer of data between processor storage and local peripheral devices. Synonymous with data channel.

I/O file. A file opened for input and output use. [OSF]

IP. Internet Protocol.

IPC. Interprocess communication.

IPL. Initial program load.

IP socket. The port concatenated with the Internet Protocol (IP) address. [OSF]

ISO/IEC. International Standards Organization/International Electrotechnical Commission.

ISPF. Interactive System Productivity Facility.

ISPF/PDF. Interactive System Productivity Facility/Program Development Facility.

item. The data in one line of an indexed field. [OSF]

J

JES. Job entry subsystem.

JES2. An MVS subsystem that receives jobs into the system, converts them to internal format, selects them for execution, processes their output, and purges them from the system. In an installation with more than one processor, each JES2 processor independently controls its job input, scheduling, and output processing. See also *JES3*.

JES3. An MVS subsystem that receives jobs into the system, converts them to internal format, selects them for execution, processes their output, and purges them from the system. In complexes that have several loosely coupled processing units, the JES3 program manages processors so that the global processor exercises centralized control over the local processors and distributes jobs to them via a common job queue. See also *JES2*.

job control. (1) Facilities for monitoring and accessing background processes. [OSF] (2) A facility that allows users to selectively stop (suspend) the execution of processes and continue (resume) their execution at a later point. The user typically employs this facility via the interactive interface jointly supplied by the terminal I/O driver and a command interpreter. Conforming implementations may optionally support job control facilities; the presence of this option is indicated to the application at compile time or run time by the definition of the `[_POSIX_JOB_CONTROL]` symbol. [POSIX.1]

job name. (1) The name of a job as identified to a system. (2) The name assigned to a JOB statement; it identifies the job to the system.

job control language (JCL). A control language used to identify a job to an operating system and to describe the job's requirements.

job entry subsystem (JES). A system facility for spooling, job queueing, and managing I/O.

job step. (1) The execution of a computer program explicitly identified by a job control statement. A job may specify that several job steps be executed. [A] (2) A unit of work represented by a single program or a procedure that contains a single program. A job consists of one or more job steps.

job step task. A task begun by an initiator/terminator in the job scheduler in accordance with specifications in an execute (EXEC) statement.

jump. In the execution of a computer program, a departure from the implicit or declared order in which instructions are being executed. [OSF]

justified. Adjacent to the indicated margin. In nonprogrammable terminals, if attribute bytes are required, left-justified information appears one character position to the right of the left margin with an unseen attribute byte in the first position. See also *justify*, *left-justify*, *right-justify*.

justify. (1) To control the printing positions of characters on a page so that both the left-hand and right-hand margins of the printing are regular. [I][A] (2) To shift the contents of a register or field so that the significant character at the specified end of the data is at the specified position. [T] (3) To align characters horizontally or vertically to fit the positioning constraints of a required format. [A] (4) To print a document with even right margins, even left margins, or both.

K

Kanji. A graphic character set consisting of symbols used in Japanese ideographic alphabets. Each symbol conveys an idea, most of which have at least two readings. Kanji is the Japanese version of Chinese ideographs. It is not an alphabet (where symbols

represent phonemes or syllables), such as Hiragana and Katakana. The Japanese government has approved a basic list of 1850 Kanji characters, but there are thousands more. See also *doublebyte character set (DBCS)*, *Hiragana*, *Katakana*.

Katakana. A character set of symbols used in one of the two common Japanese phonetic alphabets, the other being Hiragana. Each character represents a syllable. Katakana is used for phonetically writing English words in Japanese (including most data-processing terms), foreign place names, and other Japanese words of foreign origin. See also *doublebyte character set (DBCS)*, *Hiragana*, *Kanji*.

kernel. (1) The part of z/OS UNIX that contains programs for such tasks as I/O, management, and control of hardware and the scheduling of user tasks. [OSF] (2) The part of the system that is an interface with the hardware and provides services for other system layers such as system calls, file system support, and device drivers. (3) The part of an operating system that performs basic functions such as allocating hardware resources. (4) A program that can run under different operating system environments. See also *shell*. (5) A part of a program that must be in central storage in order to load other parts of the program. See also *boot* (6) Synonym for the *kernel address space*.

kernel address space. The address space containing the MVS support for z/OS UNIX services. This address space can also be called the kernel. See also *kernel*.

kernel mode. In a multiprocessor environment, the master in a master-slave relationship. The master processor operates in kernel mode, and the slave processor operates only in user mode. Kernel mode handles the interrupts and callable services. User mode informs the master when issuing a callable service. Contrast with *user mode*.

keyboard. An input device consisting of various keys that allows the user to input data, control cursor and pointer locations, and to control the dialog with the workstation. [OSF]

keyword. (1) A name or symbol that identifies a parameter. (2) A part of a processing statement or command operand that consists of a specific character string. (3) A part of a command operand that consists of a specific character string (such as `DSNAME=`). (4) A predefined word in a programming language. [OSF] (5) A reserved word. [OSF] (6) In programming languages, a lexical unit that characterizes some language construct. A keyword normally has the form of an identifier. [I]

keyword parameter. A parameter that consists of a keyword, followed by one or more values. See also *positional parameter*.

kill. An operating system command that stops a process. [OSF]

kill character. A character that deletes a line of characters entered after a prompt. [OSF]

KornShell. A command interpreter developed on UNIX, which forms the basis for the z/OS shell.

L

label. (1) In C, an identifier followed by a colon, used to identify a statement in a program. Usually the target of a **goto** or **switch** statement. [OSF] See also *labeled statement*. (2) A name in the disk or diskette table of contents that identifies a file. [OSF] See also *filename*. (3) The field of an instruction that assigns a symbolic name to the location at which the instruction begins. [OSF] (4) In programming languages, a language construction naming a statement and including an identifier. [I] (5) An identifier within or attached to a set of data elements. [T] (6) An identifier of a command generally used for branching. (7) In assembler programming, a name entry.

labeled statement. A C language statement that contains one or more identifiers followed by a colon and a statement. [OSF]

LALR. One of the many types of language parsers for context-free grammars. LALR is a look-ahead type of parser. An LR parser scans its input from left to right, constructing a rightmost derivation in reverse. The essential components of such a parser are a general driver routine, a state transition table, and a state stack. The state transition table describes the allowable actions for each state, based on the current state, current input token, and the state on top of the stack.[OSF]

LALR(1) parser. A lookahead LR parser with one token of lookahead.[OSF]

language. (1) A set of characters, conventions, and rules that is used for conveying information. [I][A] (2) In internationalization contexts, the choice of language specifies the language (for example, German, French, English) and the display format for messages and the appropriate collating sequence. [OSF]

Language Environment/370 (LE/370). A packaging concept that allows an installation to order a “combination package,” including the runtime libraries and environment to support enabled languages. LE/370 includes the Common Execution Library (CEL) and the language-specific runtime libraries.

last-in-first-out (LIFO). A queuing technique in which the next item to be retrieved is the item most recently placed in the queue. [A] Contrast with *first-in-first-out (FIFO)*.

Latin alphabet. An alphabet comprising the letters a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x,

y, and z in uppercase and lowercase, in some cases with accents on certain characters.

LE/370. Language Environment/370.

Latin Alphabet Number 1. A set of 190 graphic characters and SPACE defined in ISO 8859-1, containing the alphabets of Western European languages. This character set is identical to the character set 00697 (revised) in the IBM registry. Many corresponding national standards also specify this set. Latin Alphabet Numbers 2, 3, and 4 are defined in other parts of ISO 8859.

left. A scrolling action that displays information to the left of that currently visible in a panel body area.

left-justify. (1) To shift the contents of a register or field so that the character at the left-hand end of the data is at a specific position. [T] (2) To control the printing positions of characters on a page so that the left-hand margin of the printing is regular. [I][A]

letter. An uppercase or lowercase unit of an alphabet. [OSF]

level. (1) The degree of subordination of an item in a hierarchical arrangement. [I][A] (2) In a hierarchical database, the successive vertical dependencies of records or segments.

lexical analyzer. A program that analyzes input and breaks it into categories, such as numbers, letters, or operators.

library. (1) A collection of functions, calls, subroutines, or other data. (2) A named area on disk that can contain programs and related information (not files). A library consists of different sections, called library members. (3) A data file that contains copies of a number of individual files and control information that allows them to be accessed individually. (4) A collection of related files. For example, one line of an invoice may form an item, a complete invoice may form a file, the collection of inventory control files may form a library, and the libraries used by an organization are known as its data bank. (5) A repository for demountable recorded media, such as magnetic disk packs and magnetic tapes. [A] (6) The set of publications for a product.

library directory. In a library, an area that contains information about each member in the library; for example, the member name and the location.

licensed program (LP). (1) A software program that remains the property of the manufacturer, for which customers pay a license fee. [OSF] (2) Any separately priced program that bears an IBM copyright and is offered to customers under terms and conditions of either the Agreement for IBM Licensed Programs (ALP) or the IBM Program License Agreement (PLA). A licensed program may be a program product (PP), industry application program (IAP), field-developed

program (FDP), installed user program (IUP), or programming request for price quotation (PRPQ). (3) A program that performs a function for the user and usually interacts with and relies upon system control programming or some other IBM-provided control program. A licensed program contains logic related to the user's data and is usable or adaptable to meet that user's requirements.

LIFO (last-in-first-out). A queuing technique in which the next item to be retrieved is the item most recently placed in the queue. [A] Contrast with *first-in-first-out (FIFO)*.

line. (1) A string of characters accepted by a system as a single block of input from a terminal, such as all characters entered before a carriage return. (2) In terminal-oriented programs, a stream of bytes terminated by <newline>. (3) A sequence of text consisting of zero or more non-<newline> characters plus a terminating <newline> character. [POSIX.2] (4) A horizontal display on a screen. [OSF]

line editor. An editor that displays data one line at a time and that allows data to be accessed and modified only by entering commands. [OSF]

line mode. Synonym for *canonical mode*.

link. (1) In the file system, a connection between an inode and one or more file names associated with it. (2) Synonym for *directory entry*. (3) In data communication, a transmission medium and data link control component that together transmit data between adjacent nodes. [OSF] (4) In programming, the part of a program that passes control and parameters between separate portions of the computer program. (5) To interconnect items of data or portions of one or more computer programs: for example, the linking of object programs by a linkage editor or linking data items by pointers. [T] (6) In SNA, the combination of the link connection and the link stations joining network nodes—for example, a System/370™ channel and its associated protocols in a serial-by-bit connection under the control of Synchronous Data Link Control (SDLC).

link count. The number of directory entries that refer to a particular file. [POSIX.1]

linkage editor. (1) A computer program for creating load modules from one or more object modules or load modules by resolving cross-references among the modules and, if necessary, adjusting addresses. [T] (2) A program that resolves cross-references between separately compiled object modules and then assigns final addresses to create a single relocatable load module. If a single object module is linked, the linkage editor simply makes it relocatable. [OSF] (3) Synonymous with binder.

link-edit. To create a loadable computer program by means of a linkage editor.

link pack area (LPA). (1) An area of main storage containing reenterable routines from system libraries. Their presence in main storage saves loading time. (2) An area of virtual storage that contains reenterable routines that are loaded at IPL time and can be used concurrently by all tasks in the system.

list. (1) A sequence of one or more pipelines. (2) A data object consisting of a collection of related records.

listing. A printout, usually prepared by a language translator, that lists the source language statements and contents of a program.

literal. (1) A symbol or a quantity in a source program that is itself data, rather than a reference to data. (2) In programming languages, a unit that directly represents a value. For example, 14 represents the integer 14.

load. (1) To transfer programs or data from storage into an area of memory where the program can be run or where the data can be manipulated. [OSF] Contrast with *call*. (2) Synonym for *initial program load (IPL)*.

loader. A program that reads run files into central storage so that they can execute.

load module. (1) A computer program in a form suitable for loading into central storage for execution. [T] (2) A program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading—for example, the input to or output from an assembler, compiler, linkage editor, or execution routine. (3) The output of the linkage editor. (4) Synonym for *run file*.

local. (1) Pertaining to a device, file, or system that is accessed directly from your system, without the use of a communication line. (2) Pertaining to that which is defined and used only in one subdivision of a computer program. [T] (3) Contrast with *remote*.

locale. (1) A description of a cultural environment. [POSIX.0] (2) The definition of the subset of a user's environment that depends on language and cultural conventions. [POSIX.2] (3) A tuple generally consisting of a language, territory, and codeset specification and used in internalization configuration. [OSF]

location counter. A counter in the assembler that denotes the next byte available for code allocation. The location counter assigns storage addresses to program statements. [OSF] See also *instruction address register (IAR)*.

lock. A serialization mechanism by which a specific resource is restricted for the use of the holder of the lock.

logarithm. A mathematical operation related to the base of a numbering system. [OSF]

logger. (1) A functional unit that records events and physical conditions, usually with respect to time. [I][A]
(2) A program that enables a user entity to log in (for example, identify itself, its purpose, and time of entry) and log out with the corresponding data. This enables the appropriate accounting procedures to be carried out in accordance with the operating system.

logical mount. A mount that attaches a file system to the root directory or to a directory of another file system so that the files and directories on the file system can be referenced. The attached file system can consist of a file or many files and directories.

logical operator. A symbol that represents an operation, such as AND, OR, or NOT, on logical expressions. [OSF]

logically partitioned (LPAR) mode. A central processor complex (CPC) power-on reset mode that enables use of the PR/SM™ feature and allows an operator to allocate CPC hardware resources (including central processors, central storage, expanded storage, and channel paths) among logical partitions.

logical record. (1) A record independent of its physical environment. Portions of the same logical record may be located in different physical records, or several logical records or parts of logical records may be located in one physical record. [I] (2) A record from the standpoint of its content, function, and use rather than its physical attributes—that is, one that is defined in terms of the information it contains.

log in. (1) In UNIX systems, to gain access to a computer system by entering identification information at the workstation. (2) To begin a session at a workstation. (3) See also *log on*, the MVS term.

login. (1) In UNIX systems, the act of gaining access to a computer system by entering identification and authentication information at the workstation. (2) The unspecified activity by which a user gains access to the system. Each login shall be associated with exactly one login name. [POSIX.1] In the OpenEdition implementation, a user gains interactive access to the shell by first logging on to a TSO/E user ID through existing documented MVS externals. Once logged on to TSO/E, a user can execute the shell by entering the OMVS command.

login name. (1) A string of characters that uniquely identifies a user to the system. (2) A user name that is associated with a login. [POSIX.1]

log on. (1) In MVS TSO/E, to begin a session. (2) In SNA products, to begin a session between an application program and a logical unit (LU). (3) See also *log in*, the UNIX term.

logon. The procedure by which a terminal user begins a terminal session.

lookahead. A feature of a function that enables it to read beyond the current item to the next item and beyond. [OSF]

loop. (1) A sequence of instructions performed repeatedly until an ending condition is reached. [OSF]
(2) A closed unidirectional signal path connecting input and output devices to a system.

low-order. Least significant; rightmost. For example, in a 32-bit register (0–31), bit 31 is the low-order bit. [OSF]

LPA. Link pack area.

LPAR. Logically partitioned mode

M

machine instruction. A binary number that directs the operation of a processor. [OSF] Compilers and assembler convert source instructions to machine instructions. Synonymous with computer language.

macro. (1) A label that is declared at the start of a program or file. The label can then be used to represent the values assigned to the label in the declaration. [OSF] (2) A name or label used in place of a number of other names. [OSF] (3) The sequence of instructions or statements that a macrogenerator executes when replacing a macroinstruction. [OSF] (4) A set of statements defining the name of, format of, and conditions for generating a sequence of assembler statements from a single source statement. [OSF] (5) Synonym for *macroinstruction*. (6) See also *macroinstruction*, *routine*, *subroutine*.

macroinstruction. (1) A single instruction that, when executed, causes the execution of a predefined sequence of instructions in the same source language. [OSF] Synonymous with macro call. (2) An instruction in a source language that is to be replaced by a defined sequence of instructions in the same source language. The macroinstruction may also specify values for parameters in the instructions that are to replace it. [T] (3) An assembler language statement that causes the assembler to process a predefined set of statements called a macrodefinition. The statements normally produced from the macrodefinition replace the macroinstruction in the program. (4) Synonymous with macro.

magic number. A numeric or string constant in a file that indicates the file type.

main function. A function that has the identifier **main**. Each program must have exactly one function named **main**. This function begins and ends program execution.

mainline routine. The first subroutine encountered when link-editing. [OSF]

main program. The first program unit to receive control when a program is run. Contrast with *subprogram*.

main storage. (1) The part of internal storage into which instructions and other data must be loaded for execution or processing. [T] (2) The part of the processing unit where programs are run. (3) Program-addressable storage from which instructions and other data can be loaded directly into registers for subsequent execution or processing. [I][A] Synonymous with memory, primary storage. (4) Synonym for *central storage*. (5) See also *real storage*. Contrast with *auxiliary storage*.

makefile. A file that describes interdependencies between other files. A makefile is used in connection with the **make** command, which updates files to keep them in sync with one another. The makefile also shows the commands needed to update the files.

marker. In computer graphics, a glyph with a specified appearance that is used to identify a particular location. [I][A]

mask. A pattern of characters that controls the keeping, deleting, or testing of portions of another pattern of characters. [I][A]

master. In a multiprocessor environment, the designation for the processor that operates in kernel mode, running unparallelized code. The other processor is the slave and operates in user mode. Synonymous with master processor. See also *kernel mode*, *user mode*.

master address space. The virtual storage used by the master scheduler task.

master console. In a system with multiple consoles, the basic console used for communication between the operator and the system.

medium. The material in or on which data may be represented (for example, twisted pairs, coaxial cables, and optical fibers). [OSF]

member. (1) A data object in a structure, a union, or a library. (2) A partition of a partitioned data set. (3) A part of a partitioned data set that can be used independently of other members of the data set. (4) Synonym for *element*.

memory. (1) Storage that is an integral part of the processor unit. (2) Program-addressable storage from which instructions and other data can be loaded directly into registers for subsequent execution or processing. [OSF] (3) Storage on electronic chips. Examples of memory are random access memory, read-only memory, or registers. [OSF] (4) Synonym for *storage*. (5) In MVS, synonym for *central storage*.

message. (1) In information theory, an ordered set of characters intended to convey information. [I][A] (2) A communication sent from a person or program to another person or program. (3) Information not requested by users but presented by the computer in response to a user action or an internal process. (4) Information from the system that informs the user of a condition that may affect further processing of a current program. [OSF] (5) An error indication, or any brief information that a program writes to standard error or a queue. [OSF] (6) Information sent from one user in a multiuser operating system to another. [OSF] (7) A general method of communication between two processes. [OSF] (8) A group of characters and control bit sequences transferred as an entity. [OSF]

message queue. (1) A list of messages awaiting processing or waiting to be sent to a terminal. (2) A queue of messages within a message data set waiting to be transmitted to the host system or to a particular terminal operator.

metacharacter. (1) A character used to specify another character or series of characters. [OSF] (2) A character that may have a special meaning in a regular expression. You can usually use a backslash to remove the special meaning.

mode. (1) A method of operation. (2) A method of operation, frequently used in UNIX to refer to read, write, run, or search permissions of a file or directory. [OSF] (3) A collection of attributes that specifies a file's type and its access permissions. [POSIX.1]

model ACL. See *default ACL*.

modem (MODulator-DEMulator). (1) A device that modulates and demodulates signals transmitted over data communication facilities. (2) A device that converts digital data from a computer to an analog signal that can be transmitted on a telecommunication line, and converts the analog signal received to data for the computer.

module. (1) A discrete programming unit that usually performs a specific task or set of tasks. Modules are subroutines and calling programs that are assembled separately, then linked to make a complete program. (2) In programming languages, a language construct that consists of procedures or data declarations and that interact with other such constructs. [I] (3) A program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading; for example, the input to or output from an assembler, compiler, linkage editor, or executive routine. [A] (4) A packaged functional hardware unit designed for use with other components. [A] (5) See *load module*, *object module*.

monitor. (1) A device that observes and verifies operations of a data processing system. [OSF] (2) A functional unit that observes and records selected

activities for analysis within a data processing system. Possible uses are to show significant departures from the norm or to determine levels of utilization or particular functional units. [OSF]

mount. (1) To make a file system accessible. [OSF] (2) To logically mount a file system in another file system with the TSO/E command MOUNT. The mount point is in a directory. (3) The action taken by an NFS client to use file systems that an NFS server has made available. [OSF] (4) See also *file system*, *mount point*.

mountable file system. (1) A file system stored in an HFS data set and, therefore, able to be logically mounted in another file system. (2) Synonym for *HFS data set*.

mount point. (1) The pathname of the directory on which the file system is mounted. (2) The local directory of an NFS client where the remote directory is mounted. [OSF] (3) Either the root directory or a directory for which the *st_rdev* field of the POSIX.1 *struct stat* differs from that of its parent directory. [POSIX.2]

multiprocessing (MP). A mode of computer operation in which two or more processing units execute simultaneously under integrated control.

mutex. Mutual exclusion. A mutual exclusion lock is a read/write lock that grants access to only a single thread at any one time. [OSF] A mutex is often used to ensure that shared variables are always seen by other threads in a consistent state. Synonymous with mutex lock.

mutex lock. Synonym for *mutex*.

mutual exclusion. See *mutex*.

MVS/ESA™ System Product Version 5. Multiple Virtual Storage/Enterprise Systems Architecture.

N

named pipe. Synonym for *FIFO special file*.

national language support. The ability for users to communicate with products in the language most familiar to them.

NetView. A licensed program that lets you control, record, and automate certain operator tasks. An operator at the host can use NetView to collect, correlate, and access online data about sessions, routes, and link- or channel-attached resources.

network. (1) A collection of data processing products that are connected by communication lines for information exchange between locations. [OSF] (2) A connection between hosts that allows them to exchange information. [OSF] UNIX networks include: circuit, packet-switched, Ethernet, long-haul, short-haul, and

token-ring. (3) An interconnected group of nodes. (4) A system consisting of two or more interconnecting computing units. (5) In data processing, a user-application network.

Network File System (NFS). A protocol developed by Sun Microsystems, Inc., that allows users to directly access files on other systems in a network. [OSF] NFS provides client and server functions for distributed processing. It has transparent file sharing, lookup, and remote procedure call (RPC) interfaces.

newline. (1) A cursor-movement function that moves the cursor to the first entry field on the next line that contains an entry field. (2) The line terminator in text files and keyboard input. On the keyboard, this is generated by the Enter key. (3) See also *newline character*.

newline character (NL or <newline>). (1) A control character that causes the print or display position to move to the first position on the next line. This character is often represented by “\n”. [OSF] (2) A character that in the output stream causes printing to start at the beginning of the next line. The <newline> is the character designated by the “\n” in the C language binding. It is unspecified whether this character is the exact sequence transmitted to an output device by the system to accomplish the movement to the next line. [POSIX.2]

new-process image. A new program laid over the current program by the **exec** function or callable service.

next sequential instruction (NSI). The next logical instruction coded in a computer program.

NFS. Network File System.

NLS. National language support.

node. (1) An endpoint of a link, or a junction common to two or more links in a network. Nodes can be processors, controllers, or workstations, and they can vary in routing and other functional capabilities. [OSF] (2) In a tree structure, a point at which subordinate items of data originate. [A] (3) In SNA, an endpoint of a link or a junction common to two or more links in a network. Nodes can be distributed to host processors, communication controllers, or terminals. Nodes can vary in routing and other functional capabilities. (4) In ACF/VTAM, a point in a network defined by a symbolic name.

noncanonical mode. A **tty** input processing mode where input character erase and killing are eliminated, making input characters available to the user program as they are typed. [OSF] Synonymous with character mode, raw mode. Contrast with *canonical mode*.

nonexistent file. A file that does not physically exist on any accessible storage medium. [OSF]

NSI. Next sequential instruction.

NUL. (1) A character with all bits set to zero. [POSIX.2] (2) The **hex 00** character, used to represent the absence of a printed or displayed character. [OSF] (3) A control character used to accomplish media-fill or time-fill that can be inserted into or removed from a sequence of characters without affecting the meaning of the sequence.

null. (1) Empty. (2) Having no meaning. (3) Having no value. [OSF] (4) See also *NUL*, *NULL*.

NULL. In C language, a pointer that does not point to a data object.

null character. (1) A character with all bits set to zero [POSIX.2]. (2) See *NUL*.

null signal. A signal parameter of 0 (zero). [OSF]

null string. (1) A string containing no element. [T] (2) The notion of a string depleted of its entities, or the notion of a string prior to establishing its entities. [A] (3) Synonymous with empty string, null character string.

null-terminated. Having a zero byte at the end. In the C language, character strings are stored this way internally.

null value. (1) A parameter position for which no value is specified. (2) When used with a relational data base, an indication that no data value has been assigned to the intersection of a row and a column in a relational table.

numeric. Pertaining to any of the digits 0 through 9.

numeric constant. (1) An integer or complex constant. (2) A constant that expresses an integer, real, or complex number. [OSF]

O

object. (1) An autonomous data item that is an abstraction of some idea or process. Objects hide their implementation to the manipulating program or function by providing methods. (2) One of a set of items that actions work on.

object code. (1) Output from a compiler or assembler which is itself executable machine code or is suitable for processing to produce executable machine code. [A] Contrast with *source code*. (2) Machine-executable instructions, usually generated by a compiler from source code written in a higher level language (such as C language). For programs that must be linked, object code consists of relocatable machine code. [OSF]

object file. (1) A member file in an object library. (2) A regular file containing the output of a compiler, formatted as input to a linkage editor for linking with other object files into an executable form. The methods

of linking are unspecified and may involve the dynamic linking of objects at run time. The internal format of an object file is unspecified, but a conforming application does not assume an object file is a text file. [POSIX.2]

object library. An area on a direct access storage device used to store object programs and routines.

object module. (1) A set of instructions in machine language produced by a compiler from a source program. (2) A portion of an object program suitable as input to a linkage editor. [T] (3) See also *module*.

object program. A fully compiled or assembled program that is ready to be loaded into the computer. [I][A]

octal. (1) A base-eight numbering system. [OSF] (2) Pertaining to a fixed-radix numeration having a radix of eight. [I][A]

OMVS group ID. See *group ID (GID)*.

OMVS segment. The portion of a RACF profile that contains OMVS logon information.

OMVS user ID. See *user ID*.

open. (1) Conforming to industrywide standards. (2) To make a file available to a program for processing. [OSF] Contrast with *close*.

open file. A file that is currently associated with a file descriptor. [POSIX.1]

open system. (1) A system whose characteristics comply with standards made available throughout the industry and that therefore can be connected to other systems complying with the same standards. [T] (2) A comprehensive and consistent set of international information technology standards and functional standards profiles that specify interfaces, services, and supporting formats to accomplish interoperability and portability of applications, data and people. [T] (3) In computer security, a system in which resources that are not defined to the system are not protected. Contrast with *closed system*. (4) A system that implements sufficient open specifications for interfaces, services, and supporting formats to enable properly engineered applications software to be ported with minimal changes across a wide range of systems, to interoperate with other applications on local and remote systems, and to interact with users in a style that facilitates user portability. [POSIX.0] (5) A system that uses standard-based communication to work with other such systems to accomplish business tasks. (6) A system that is part of other such systems in an enterprise in which the following criteria apply:

- All data within each system is accessible by any user or application (within the limits of some security feature).

- Any application within the enterprise can be run from a user's terminal without the user or the program that calls the application needing to specify where the application is or how it works.
- Software or hardware system configurations can be acquired from more than a single vendor. Products from one vendor can communicate with products with products from other vendors.
- Applications are portable from system to system, independently of the differences in hardware.
- Data is portable from system to system.
- New applications and systems integrate well with the current applications and systems.
- All systems conform to standards formalized by international or national standards committees or consortia, rather than to standards developed and promulgated by hardware or software vendors.
- All systems use standards-based UNIX.

open system application program interface. A combination of standards-based interfaces specifying a complete interface between an application program and the underlying application platform. [POSIX.0]

open system environment (OSE). The comprehensive set of interfaces, services, and supporting formats, plus user aspects for interoperability or for portability of applications, data, or people, as specified by information technology standards and profiles. [POSIX.0]

operand. (1) An argument to a command that is generally used as an object supplying information to a utility necessary to complete its processing. Operands generally follow the options in a command line. [POSIX.2] (2) An instruction field that represents data (or the location of data) to be manipulated or operated upon. Not all instructions require an operand field. [OSF] (3) An identifier, constant, or expression that is grouped with an operator. [OSF] (4) An entity on which an operation is performed. [T][A] (5) Information entered with a command name that defines the data on which a command processor operates and that controls the execution of the command processor. (6) Information entered with a macro instruction. (7) See also *keyword*, *parameter*.

operating system. (1) Software that controls the execution of programs. An operating system may provide services such as resource allocation, scheduling, input/output control, and data management. [I][A] (2) A set of programs that control how the system works. [OSF] (3) The software that deals with the most basic operations that a computer does. For example, when a normal program wants to perform a read or write operation, the program almost never does the I/O itself; instead, it asks the operating system to perform the I/O. The operating system can then make sure that the I/O is done in a safe and correct way, to prevent the loss or damage of files and directories.

Operating systems often deal with the problems of coordinating several jobs or users who want to use a computer simultaneously.

operation. (1) A specific action (such as add, multiply, or shift) that the computer performs when requested. [OSF] (2) A set of step-by-step actions specified by a procedure, function, or routine. [OSF] (3) The task performed by a given routine or procedure. [OSF]

operator. (1) A person who operates a device. (2) The person responsible for monitoring the system console and controlling the system. (3) A symbol (such as +, −, or *) that represents an operation (in this case, addition, subtraction, multiplication). [OSF] (4) A symbol that represents an operation to be done. (5) See *logical operator*.

optimize. To improve the speed of a program or to reduce the use of storage during processing. [OSF]

option. (1) A specification in a statement that can influence the execution of the statement. (2) An argument to a command that is generally used to specify changes in the *utility's* default behavior. [POSIX.2] (3) Synonym for *process option*.

orphaned process group. A process group in which the parent of every member is either itself a member of the group or is not a member of the group's session. [POSIX.1]

OSE. Open system environment.

other class. The processes that have access authority to protected resources but that are not in the owner class or group class.

output. (1) The result of processing data. [OSF] (2) Pertaining to a functional unit or channel involved in an output process, or to the data or involved in such a process. (3) Data transferred from storage to an output device.

output device. Synonym for *output unit*.

output file. (1) A file that a program opens so that it can write to that file. (2) A file that contains the results of processing.

output list. A list of variables from which values are written to a file or device.

output redirection. The specification of an output destination other than the standard one. [OSF]

output stream. (1) Diagnostic messages and other output data issued by an operating system or a processing program on output devices especially activated for this purpose by the operator. [OSF] (2) Messages and other output data that an operating system or a processing program displays on output devices.

output unit. A device in a data processing system that can receive data from the system. Synonymous with output device.

overflow. (1) That portion of an operation's result that exceeds the capacity of the intended unit of storage. (2) In a register, the loss of one or more of the leftmost whole-number digits because the result of an operation exceeded the size of the register. [OSF]

overflow exception. A condition caused by the result of an arithmetic operation having a magnitude that exceeds the largest possible number.

overlay. (1) To load a segment of code or data into storage that had been occupied by other code or data that is currently not needed. (2) To write over (and thereby destroy) an existing file. [OSF] (3) A program segment that is loaded into central storage, replacing all or part of a previously loaded program segment. (4) Repeatedly using the same areas of internal storage during different states of a program. [OSF]

override. (1) A parameter or value that replaces a previous parameter or value. (2) To replace a parameter or value.

overwrite. To write into an area of storage, thereby destroying the data previously stored in the same area. [T]

owner. The user who has the highest level of access authority to a data object or action, as defined by the object or action. [OSF]

P

P1003.2. POSIX.2

package. (1) For the loader, a collection of object entities that share a common name space. Symbol names are unique within a package. Symbols from different packages may bear identical symbol names, because they are distinguished by their package names. [OSF] (2) A specified group of related object management classes, denoted by an object identifier. [OSF]

padding. Bytes inserted in the data stream to maintain alignment of the protocol requests on natural boundaries. Padding increases the ease of portability to some machine architectures. [OSF]

page. (1) A block of instructions, data, or both. (2) In a virtual storage system, a fixed-length block that has a virtual address and is transferred as a unit between real storage and auxiliary storage.

page frame. (1) In real storage, a storage location having the size of a page. [I][A] (2) An area of main storage used to hold a page. [A]

parallel processing. The simultaneous processing of units of work by many servers. The units of work can be either transactions or subdivisions of large units of work (batch).

Parallel Sysplex®. A sysplex that uses one or more coupling facilities.

parameter. (1) A variable that is given a constant value for a specified application and that may denote the application. [I][A] (2) A name in a procedure that is used to refer to an argument passed to that procedure. (3) An argument the user supplies to a command or function. [OSF] (4) Data passed between programs or procedures. (5) See also *operand*.

parent. See *parent directory*, *parent environment*, *parent process*, *parent segment*.

parent directory. (1) The directory one level above the current directory. (2) When discussing a given directory, the directory that both contains a directory entry for the given directory and is represented by the pathname dot-dot in the given directory. [POSIX.1] (3) When discussing other types of files, a directory containing a directory entry for the file under discussion. [POSIX.1]

parent process. A process created to carry out a program. The parent process in turn creates child processes to execute requests. Contrast with *child process*. See also *parent process ID*, *process*.

parent process ID. An attribute of a new process after it is created by a currently active process. The parent process ID of a process is the process ID of its creator, for the lifetime of the creator. After the creator's lifetime has ended, the parent process ID is the process ID of an implementation-defined system process. [POSIX.1] In the z/OS UNIX implementation, the parent process ID of the children of an ended process is set to the process ID of the INIT process, or 1.

parent segment. In a hierarchical database, a segment that has one or more dependent segments below it in the hierarchy. See also *child segment*.

parity. The state of being either even-numbered or odd-numbered. [OSF]

parity check. A test to determine whether the number of ones (or zeros) in an array of binary digits is odd or even.

parmlib. All the members in the SYS1.PARMLIB PDS that contain parameters setting the limits and controlling the behavior of z/OS.

parmlib member. One of the members in the SYS1.PARMLIB PDS that contain parameters setting the limits and controlling the behavior of z/OS.

parse. To analyze the operands entered with a command and build a parameter list for the command processor from the information.

parser. A program that interprets user input and determines what to do with the input.

partitioned data set (PDS). A data set in direct access storage that is divided into partitions, called members, each of which can contain a program, part of a program, or data. Synonymous with program library. Contrast with *sequential data set*.

passwd. A file in the */etc* directory that contains the names of people who can use the system, the name of the shell, and the name of the **HOME** directory.

path. (1) In a network, any route between any two nodes. [T] (2) In a database, a sequence of segment occurrences from the root segment to an individual segment.

PATH. An environment variable that lists which directories on the disk need to be searched for a command. z/OS shells and utilities binaries should be placed in a directory on this search list.

pathname. (1) A filename specifying all directories leading to the file. (2) See also *relative pathname*. (3) A filename specifying all directories leading to a file plus the filename itself. (4) A string that is used to identify a file. A pathname consists of, at most, [PATH_MAX] bytes, including the terminating null character. It has an optional beginning slash, followed by zero or more filenames separated by slashes. If the pathname refers to a directory, it may also have one or more trailing slashes. Multiple successive slashes are considered to be the same as one slash. A pathname that begins with two successive slashes may be interpreted in an implementation-defined manner, although more than two leading slashes shall be treated as a single slash. [POSIX.1] In the z/OS UNIX implementation, the C/370 functions **fopen()**, **freopen()**, **remove()**, and **rename()** interpret names with exactly two leading slashes, no leading blanks or other characters, and the third character not a slash to mean that the rest of the name refers to a traditional MVS data set. (5) See also *absolute pathname*, *relative pathname*.

pathname component. Synonym for *filename*.

path prefix. A pathname, with an optional ending slash, that refers to a directory. [POSIX.1]

pattern. (1) A regular expression or series of regular expressions that define the search pattern. (2) A sequence of characters used either with regular expression notation or for pathname expansion, as a means of selecting various character strings or pathnames, respectively. The syntaxes of the two patterns are similar, but not identical; the standard always indicates the type of pattern being referred to in the immediate context of the use of the term.

[POSIX.2] (3) A sequence of characters used by commands that search for strings. Some characters have special meanings in patterns; for example, \$ stands for the end of a line and *abc\$* refers to the sequence *abc* appearing at the end of a line. Some patterns can be matched by many different strings.

pattern matching. (1) The identifying of one of a predetermined set of items which has the closest resemblance to a given object, by comparing its coded representation against the representation of all the items. [T] (2) Specifying a pattern of characters that the system should find. [OSF] (3) The process of searching for strings of characters that conform to the pattern of characters in a regular expression.

PDS. Partitioned data set.

pending. Waiting, as in an operation that is pending. [OSF]

performance. A measure of a computer system or subsystem to perform its functions—for example, response time, throughput, number of transactions per second. [POSIX.0]

period. The character “.” The term *period* is contrasted against *dot*, which is used to describe a specific directory entry. See also *dot*. [POSIX.2]

permanent storage. (1) Direct-access storage. (2) Fixed storage. (3) A storage device whose contents cannot be modified. [I][A]

permission. (1) A code that determine how the file can be used by any users who work on the system. [OSF] (2) The modes of access to a protected object. [OSF]

permission bits. The POSIX-defined owner, group, and other permission bits that are stored with an HFS object.

PF (programmed function) key. A key on the keyboard of a display device that passes a signal to a program to call for a particular program operation.

PGID. Process group ID.

phase. A distinct part of a process in which related operations are performed.

physical file. A database file that describes how data is to be presented or received from a program and how data is actually stored in the database. A physical file contains one record format and one or more members.

physical file system (PFS). The part of the operating system that handles the actual storage and manipulation of data on a storage medium. A PFS must adhere to the interface described in *z/OS UNIX System Services File System Interface Reference*. Also called an *installable file system*.

PID. Process ID.

pipe. (1) An interprocess communication mechanism that connects an output file descriptor to an input file descriptor. Usually the standard output of one process is connected to the standard input of another, forming a pipeline. (2) A sequence of one or more commands in FIFO order. The output of one command becomes the input to the next command. A pipe usually contains several filters. Pipes allow related or unrelated processes to read and write to each other as if they were files; they allow unidirectional communication from one process to another. z/OS UNIX treats pipes as though they were files. A named pipe has a directory name and is accessed by a pathname. An unnamed pipes must be used between a parent process and a child process. (3) An object accessed by one of the pair of file descriptors created by the **pipe()** function. Once a pipe is created, the file descriptors can be used to manipulate it, and it behaves identically to a FIFO special file when accessed in this way. It has no name in the file hierarchy. [POSIX.1] (4) To direct data so that the output from one process becomes the input to another process. (5) An I/O stream that has a descriptor and can be used in unidirectional communications between related processes. [OSF] (6) See also *filter*, *pipeline*.

pipeline. (1) A chain of two or more processes connected by pipes. Each process in the chain acts as a filter, reading data from the standard input, performing some transformation, and writing the results to the standard output. (2) A direct, one-way connection between two or more processes. (3) A serial arrangement of processors or a serial arrangement of registers within a processor. Each processor or register performs part of a task and passes results to the next processor. Several parts of different tasks can be performed at the same time. (4) To perform processes in a series. (5) For increased processing speed, to start execution of an instruction sequence before the previous instruction sequence is completed.

pointer. (1) In the C language, a variable that holds the address of a data object or a function. (2) A physical or symbolic identifier of a unique target. (3) An identifier that indicates the location of an item of data. [A] (4) A data element that indicates the location of another data element. [T] (5) In computer graphics, a manually operated functional unit used to specify an addressable point. A pointer may be used to conduct interactive graphic operations, such as selection of one member of a predetermined set of display elements, or indication of a position on a display space while generating coordinate data. [T] (6) The symbol displayed on the screen that is moved by a pointing device, such as a mouse. The pointer is used to point at items users can select.

pointing device. An instrument, such as a mouse, trackball, or joystick, used to move a pointer on the screen.

polling. Interrogation of devices for such purposes so as to avoid contention, to determine operational status, or to determine readiness to send or receive data. [A]

port. (1) To move a program or operating system from one computer or hardware configuration to another without modifying it. (2) To make the programming changes necessary to allow a program that runs on one type of computer to run on another type of computer. (3) An entrance to or exit from a network.

portability. (1) The ability to run a program on more than one computer without modifying it. (2) The ability to use applications, data sets, or files with different operating systems. (3) The ease with which software can be transferred from one information system to another. [POSIX.0]

portable character set. The set of characters described in POSIX.2 2.4 that is supported on all conforming systems. See also *portable filename character set*. [POSIX.2]

portable filename character set. The set of characters from which portable filenames are constructed. For a filename to be portable across conforming implementations of POSIX.1, it shall consist only of the uppercase and lowercase characters of the alphabet (A through Z and a through z), the digits 0 through 9, the period (.), the underscore (_), and the hyphen (-). The hyphen shall not be used as the first character of a portable filename. Uppercase and lowercase letters shall retain their unique identities between conforming implementations. In the case of a portable pathname, the slash character may also be used. [POSIX.1]

Portable Operating System Interface. See *POSIX*.

port number. The part of a socket address that identifies a port within a host.

position. (1) Any location in a string that may be occupied by an element and that is identified by a serial number. [T] (2) The location of a character in a series, as in a record, a displayed message, or a computer printout.

positional parameter. (1) A shell facility that assigns values from the command line to variables in a program. [OSF] (2) A parameter that must appear in a specified location relative to other positional parameters. (3) One of the command line arguments to a shell file. Positional parameters are referenced by \$1, \$2, and so on.

POSIX. Portable Operating System Interface for Computer Environments, an interface standard governed by the IEEE and based on UNIX. POSIX is not a product. Rather, 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.

POSIX open system environment (POSIX OSE). The open system environment in which the standards included are not in conflict with ISO/IEC 9945 and are:

- International Standards and Profiles (developed by ISO, IEC, or CCITT)
- Regional Standards and Profiles (developed by a group recognized as an official body by a regional governmental entity, such as the European Community)
- National Information Technology Standards and Profiles (developed by a national standards body recognized as such by ISO, IEC, or CCITT, as appropriate)

[POSIX.0]

PPID. Parent process ID.

precedence. (1) The priority system for grouping different types of operators with their operands. [OSF] (2) In programming languages, an order relation defining the sequence of the application of operators within an expression. [OSF]

precision. (1) A measure of the ability to distinguish between nearly equal values. See also *single-precision*, *double-precision*. (2) The degree of discrimination with which a quantity is stated. For example, a three-digit numeral discriminates among 1000 possibilities. (3) The number of digits that are printed or displayed.

preprocessor. (1) A program that examines the source program for preprocessor statements, which are then executed, resulting in the alteration of the source program. (2) In emulation, a program that converts data from the format of an emulated system to the format accepted by an emulator. (3) A functional unit that effects preparatory computation or organization. [T]

primary. An irreducible unit of data. For example, a single constant, variable, or array element.

primary address space. The address space whose segment table is used to fetch instructions.

printable character. One of the characters included in the **print** character classification of the LC_CTYPE category in the current locale. [POSIX.2]

print file. A file that is created for the purpose of printing data.

privilege . See *appropriate privileges*.

privileged user. A user logged into an account with root user authority.

procedure. (1) A sequenced set of statements that may be used at one or more points in one or more computer programs, and that usually has one or more input parameters and yields one or more output parameters. [T] (2) The description of the course of action taken for the solution of a problem. [A] (3) In a programming language, a block, with or without formal

parameters, whose execution is invoked by means of a procedure call. [I] (4) A set of related control statements that cause one or more programs to be executed. (5) Synonym for *function*.

process. (1) A function being performed or waiting to be performed. (2) An executing function, or one waiting to execute. (3) A function, created by a **fork()** request, with three logical sections:

- Text, which is the function's instructions.
- Data, which the instructions use but do not change.
- Stack, which is a push-down, pop-up save area of the dynamic data that the function operates upon.

(4) A program using z/OS UNIX services. The program can be created by a **fork()** function or fork callable service, or the program can be dubbed because it requested z/OS UNIX services. The three types of processes are:

- User processes, which are associated with a user at a workstation
- Daemon processes, which do systemwide functions in user mode, such as printer spooling
- Kernel processes, which do systemwide functions in kernel mode, such as paging

A process can run in a user address space, a forked address space, or a kernel address space. In an MVS system, a process is handled like a task. See also *task*. (5) An address space and one or more threads of control that execute within that address space, and their required system resources. [POSIX.0] (6) An address space and single thread of control that executes within that address space, and its required system resources. A process is created by another process issuing the **fork()** function. The process that issues **fork()** is known as the parent process, and the new process created by the **fork()** is known as the child process. [POSIX.1] (7) A sequence of actions required to produce a desired result. [OSF] (8) An entity receiving a portion of the processor's time for executing a program. [OSF] (9) An activity within the system that is started by a command, a shell program, or another process. Any running program is a process. (10) A unique, finite course of events defined by its purpose or by its effect, achieved under given conditions. (11) Any operation or combination of operations on data. (12) The current state of a program that is running—including a memory image, the program data, the variables used, the general register values, the status of opened files used, and the current directory. Programs running in a process must be either operating system programs or user programs. [OSF] (13) A running program, including the memory occupied, the open files, the environment, and other attributes specific to a running program.

process accounting. An analysis of how each process uses the processing unit, memory, and I/O resources. [OSF]

process check. Synonym for *program check*.

process group. A collection of processes that permits the signaling of related processes. Each process in the system is a member of a process group that is identified by a process group ID. A newly created process joins the process group of its creator. [POSIX.1]

process group ID (PGID). The unique identifier representing a process group during its lifetime. A process group ID is a positive integer that can be contained in a *pid_t*. It shall not be reused by the system until the process group lifetime ends. [POSIX.1]

process group leader. A process whose process ID is the same as its process group ID. [POSIX.1]

process ID (PID). (1) A unique number assigned to a process that is running. [OSF] (2) The unique identifier representing a process. A process ID is a positive integer that can be contained in a *pid_t*. A process ID shall not be reused by the system until the process lifetime ends. In addition, if there exists a process group whose process group ID is equal to that process ID, the process ID shall not be reused by the system until the process group lifetime ends. A process that is not a system process shall not have a process ID of 1. [POSIX.1]

The equivalent in MVS is an address space identifier (ASID).

process image. See *new-process image*.

processing. (1) The performance of logical operations and calculations on data, including temporary retention of data in processor storage while the data is being operated on. (2) The action of performing operations on input data.

processor. (1) In a computer, a functional unit that interprets and executes instructions. A processor consists of at least an instruction control unit and an arithmetic and logic unit. [T] (2) A functional unit, part of another unit such as a terminal or a processing unit, that interprets and executes instructions.

production system. A system where application programs that are already developed and tested run on a regular basis.

profile. (1) A file containing customized settings for a system or user. [OSF] (2) Data that describes the significant characteristics of a user, a group of users, or one or more computer resources. (3) A set of instructions to initialize a user's shell session. The shell automatically reads and executes these commands if this file is in the user's home directory. (4) In computer security, a description of the characteristics of an entity to which access is controlled. (5) A set of one or more base standards, and, where applicable, the identification of chosen classes, subsets, options, and parameters of those base standards, necessary for accomplishing a particular function. [POSIX.0] (6) See also *ENV*.

program. (1) A prepared sequence of instructions to the system to accomplish a defined task. The term *program* in POSIX.2 encompasses applications written in the shell command language, complex utility input languages (for example, **awk**, **lex**, **sed**, and so on), and high-level languages. [POSIX.2] (2) A sequence of instructions suitable for processing by a computer. Processing may include the use of an assembler, a compiler, an interpreter, or a translator to prepare the program for execution, as well as to execute it. [I] (3) In programming languages, a logical assembly of one or more interrelated modules. [I] (4) A sequence of instructions that a computer can interpret and execute.

program CCSID. For Enhanced ASCII functionality, a 16-bit value that identifies the current character set of text strings within a program. This is to be stored in the file tag of new files or used for the automatic conversion of old files when automatic conversion is in effect. It is the value that was stored in *ThliCcsid*. See also *automatic conversion* and *file tag*.

program check. A condition that occurs when programming errors are detected by a processor or an I/O channel. A program check results from execution of a program that improperly specifies or uses instructions, operands, or control information. Synonymous with process check, process exception, program exception (PROGX).

program control. A RACF function that enables an installation to control who can run RACF-controlled programs.

program counter. Synonym for *instruction address register (IAR)*.

program exception (PROGX). Synonym for *program check*.

program status word (PSW). An area of storage used to indicate the order in which instructions are executed, and to hold and indicate the status of the computer system. Synonymous with processor status word.

program text. The executable part of a program. [OSF] See also *text*.

program unit. A main program or a subprogram. [OSF] Synonymous with *module*.

prolog. A user-written description of definitions that have been defined for an application program, record, or table. A prolog is used for documentation.

prompt. (1) A displayed symbol or message that requests information or operator action. [OSF] (2) A message issued to a terminal user requesting information necessary to continue processing. (3) A common action that users request while the cursor is in an entry field. A prompt produces a menu of available

choices for that entry field. Users can select a choice from the menu to insert in the entry field.

protection. An arrangement for restricting access to or use of all or part of a computer system.

protocol. (1) A set of semantic and syntactic rules that determine the behavior of entities in performing communication functions. [POSIX.0] (2) A specification for the format and relative timing of information exchanged between communicating parties. (3) A set of semantic and syntactic rules that determines the behavior of functional units in achieving communication. [I] (4) In SNA, the meanings of, and the sequencing rules for, requests and responses used for managing the network, transferring data, and synchronizing the states of network components.

pseudoterminal (pty). A special file in the `/dev` directory that effectively functions as a keyboard and display device. Synonymous with pseudo-TTY.

pseudo-TTY. Synonym for *pseudoterminal*.

PSW. Program status word

pthread. See *thread*.

pty. Pseudoterminal.

Q

qualified name. (1) A data name explicitly accompanied by a specification of the class to which it belongs in a specified classification system. [I][A] (2) A name that has been made unique by the addition of one or more qualifiers. See also *qualifier*.

qualifier. (1) A unique name used to identify another name. [OSF] (2) A modifier that makes a name unique. (3) All names in a qualified name other than the rightmost, which is called the simple name.

query. (1) In interactive systems, an operation at a terminal that elicits a response from the system. (2) The action of searching data for desired information. [OSF] (3) In data communication, the process by which a master station asks a slave station to identify itself and to give its status. [OSF] (4) A request for information from a file based on specific conditions.

queue. (1) A line or list formed by items in a system waiting for service—for example, batched jobs to be executed, tasks to be performed, or messages to be transmitted in a message switching system. (2) One of a pair of waiting lines in a module that processes a stream (a full-duplex connection between a user process and a device). There is one queue for each direction. The queue consists of the data queue itself, a put procedure (which places messages on the data queue), a service procedure, and status information. (3) To arrange in or form a queue.

quiesce. To end a process by allowing operations to complete normally.

quote. To mask the special meaning of certain characters, causing them to be taken literally. [OSF]

R

RACF. Resource Access Control Facility.

random processing. (1) The processing of records according to an order other than their order in a file. (2) The treatment of data without respect to its location in external storage, and in an arbitrary sequence governed by the input against which it is to be processed. (3) Contrast with *sequential processing*.

raw mode. Synonym for *noncanonical mode*.

read lock. A lock that prevents any other process from setting a write lock on any part of the protected area. [OSF] Contrast with *write lock*. See also *lock*.

read-only file. In file system mounting, a file whose data can be read but not copied, printed, or modified. Contrast with *editable file version*.

read-only file system. A file system that has implementation-defined characteristics restricting modifications. [POSIX.1] In z/OS UNIX, it is a file system specified as read-only on the MOUNT command or parmlib statement that mounted the file system. No updates are made or allowed to a read-only file system. Writes are permitted to FIFO special files within a read-only file system, but FIFO data is kept only in memory and no updates are made to the disk.

real group ID. (1) For each user, any group ID defined in the password file. [OSF] (2) The attribute of a process that, at the time of process creation, identifies the group of the user who created the process. See also *group ID*. [POSIX.1]

real storage. The main storage in a virtual storage system. Physically, real storage and main storage are identical. Conceptually, however, real storage represents only part of the range of addresses available to the user of a virtual storage system. Traditionally, the total range of addresses available to the user was that provided by main storage. [I][A] Synonymous with real memory.

real user ID. (1) For each user, the user ID that is specified in the `/etc/passwd` file. [OSF] (2) The attribute of a process that, at the time of process creation, identifies the group of the user who created the process. See also *user ID*. [POSIX.1]

reason code. A value used to indicate the specific reason for an event or condition.

record. (1) In programming languages, an aggregate that consists of data objects, possibly with different attributes, that usually have identifiers attached to them.

[I] (2) A set of data treated as a unit. [T] (3) A collection of fields treated as a unit. [OSF] (4) A self-contained collection of information about a single object. A record is made up of a number of distinct items, called fields. A number of shell programs (for example, **awk**, **join**, and **sort**) are designed to process data consisting of records separated by newlines, where each record contains a number of fields separated by spaces or some other character. **awk** can also handle records separated by characters other than newlines. (5) Using a function to define itself. [OSF] (6) See *fixed-length record*, *variable-length record*.

record name. A data name for a record described in a record description entry.

recovery procedure. (1) An action performed by the user when an error message appears on the display screen. This action usually permits the program to continue or permits the user to run the next job. [OSF] (2) The method of returning the system to the point where a major system error occurred and running the recent critical jobs again. [OSF] (3) See also *recovery routine*.

recovery routine. A routine entered when an error occurs during the performance of an associated operation. It isolates the error, assesses the extent of the error, indicates subsequent action, and attempts to correct the error and resume operation. See also *recovery procedure*.

recursion. (1) The performance of an operation in several steps, with each step using the output of the preceding step. (2) Using a function to define itself. [OSF] (3) Recursion.

recursive. (1) Pertaining to a process in which each step makes use of the results of earlier steps. (2) Pertaining to a program or routine that calls itself after each run until it is interrupted or until a specified condition is met.

redirect. To divert data from a process to a file or device to which it would not normally go. [OSF]

redirection. (1) A system profile construction method of starting at a base platform and adding new services by allowing a service component to ask the base platform to redirect all requests for that type of service to the service component. [POSIX.0] (2) Changing the association between files and file descriptors for a program. A process inherits file descriptors from the process that created the program (usually the shell). A file descriptor's standard input and standard output are usually associated with the keyboard and display screen, respectively. The shell can arrange for these descriptors (or any others) to be associated with other files before creating the new process. In particular, `<infile` redirects the standard input from **infile**, while `>outfile` redirects the standard output to **outfile**.

reenterable. Deprecated term for *reentrant*.

reentrant. The attribute of a program or routine that allows the same copy of the program or routine to be used concurrently by two or more tasks. Synonymous with *reenterable*.

reentrant program. A program that can be used by more than one computer program simultaneously.

refer. A bibliographic preprocessor for **nroff** and **troff**. Refer is designed for literature citations, and it supports data entry, indexing, sorting, retrieval, printing, citations, and either footnote or end note numbering. [OSF]

region. (1) A contiguous area of virtual storage that have common characteristics and that can be shared between processes. (2) An area within a bitmap, a pixmap, or a screen.

register. (1) A storage device having a specified storage capacity such as a bit, byte, or computer word, and usually intended for a special purpose. [T] (2) See *general-purpose register (GPI)*.

regular expression (RE). (1) A pattern (sequence of characters or symbols) constructed according to the rules defined in POSIX.2 2.8. [POSIX.2] (2) A set of characters, metacharacters, and operators that define a string or group of strings in a search pattern. [OSF] (3) A string containing wildcard characters and operations that define a set of one or more possible strings. [OSF] (4) A more technical term for *pattern*. (5) See also *wildcard character*.

regular file. A file that is a randomly accessible sequence of bytes, with no further structure imposed by the system. [POSIX.1]

reject. To cause portions of applied updates from becoming permanent parts of the product, based on the results of a test period. Contrast with *commit*.

relational operator. (1) An operator that operates on at least two operands and yields a truth value. For example, `≤` (as in `5≤3`, yielding false), "is initial word of" (as in `XYZ` is initial word of `XYZU`, yielding true). [T] (2) The reserved words or symbols used to express a relational condition or a relational expression.

relative pathname. (1) The name of a directory or file expressed as a sequence of directories followed by a filename, beginning from the current directory. Relative pathnames do not begin with a `/` (slash) but are relative to the current directory. (2) A pathname that does not begin with a slash. The predecessor of the first filename in the pathname is taken to be the current working directory of the process. [POSIX.1] See also *absolute pathname*.

relative record number (RRN). (1) A number that expresses the location of a record in relation to a base position in the file containing it. (2) A number that indicates the location of a logical record, expressed as a

difference with respect to a base address. The RRN is used to retrieve that logical record from the data set. Synonymous with *slot*.

remote. (1) Pertaining to a system or device that is accessed through a communication line. [OSF] (2) Pertaining to a system, program, or device that is accessed through a telecommunication line; synonymous with *link-attached*. Contrast with *local*.

remote terminal. A terminal attached to a system through a data link. [OSF]

reply. A response to an inquiry.

request. (1) A statement used to send an output record to the terminal or receive an input record from the terminal. (2) A message unit that signals initiation of a particular action or protocol. (3) A directive, by means of a basic transmission unit, from an access method that causes the network control program to perform a data-transfer operation or auxiliary operation.

requester. A display station or interactive communication session that requests a program to be run. [OSF]

reset. (1) To cause a counter to take the state corresponding to a specified initial number. [I][A] (2) To put all or part of a data processing device back to a prescribed state. [I][A] (3) On a virtual circuit, reinitialization of data flow control. (4) To return a device or circuit to a clear state. (5) In X.25 communication, to reinitialize the flow of control on a virtual circuit, which eliminates all data that may be in transit for the virtual circuit at the time of resetting. [OSF]

resource. Any facility of the computing system or operating system required by a job or task, including main storage, I/O devices, the processing unit, data sets, control or processing programs, execution time, execution levels, locks, and queues.

Resource Access Control Facility (RACF). An IBM-licensed program that provides for access control by identifying and by verifying the users to the system, authorizing access to protected resources, and logging the detected unauthorized access to protected resources.

response. An answer to an inquiry.

restore. To return to an original value or image. For example, to restore a library from diskette. See also *dump*.

restricted shell. (1) A security feature that provides a controlled shell environment with limited features. [OSF] (2) A subset of the Bourne shell that provides restrictions to ensure system integrity.

result. An entity produced by an operation. [OSF]

resume. An action that returns users to the application and the current panel body.

retrieve. (1) To locate data in storage and read it so that it can be processed, printed, or displayed. Contrast with *store*. (2) A common action requested only while the command area is visible. It redisplay, one at a time, the previous commands that were issued. Each previous command appears in the command area entry field. (3) Contrast with *store*.

retry. To try again the operation that caused the device error message. [OSF]

return code. (1) A code used to influence the execution of succeeding instructions. [A] (2) A value returned to a program to indicate the results of an operation requested by that program. (3) Synonymous with *return value*.

return statement. A C language control statement that contains the word *return* followed by an optional expression and a semicolon. [OSF]

return value. (1) A value returned to a program to indicate the results of an operation requested by that program. (2) Synonym for *return code*.

right-justify. (1) To control the positions of characters on a page so that the right-hand margin of the printing is regular. [I][A] (2) To shift the contents of a register or field so that the character at the right-hand end of the data is at a specific position. [T] (3) To align characters horizontally so that the rightmost character of a string is in a specified position. [A]

root. (1) The starting point of the file system. (2) The first directory in the system. (3) The user name for the system user with the most authority.

root authority. The user name for the system user with the most authority. Synonymous with *superuser*.

root directory. (1) The directory that contains all other directories in the system. (2) The lowest directory in the file system hierarchy. It is referred to as *"r"*. (3) A directory, associated with a process, that is used in pathname resolution for pathnames that begin with a slash. [POSIX.1] (4) The top directory in the file system tree. UNIX and POSIX-conforming systems have a single root directory, with mounted devices. (5) See also *effective root directory*.

root file system. The basic file system, onto which all other file systems can be mounted. The root file system contains the operating system files that get the rest of the system running.

routine. (1) A program or sequence of instructions called by a program that may have some general or (2) A set of statements in a program causing the system to perform an operation or a series of related operations. [OSF] (3) See also *macro*, *subroutine*.

row. A horizontal arrangement of characters or other expressions. [A] Contrast with *column*.

RRN. Relative record number.

RTLS. Run-Time Library Services. This service enables you to access multiple levels of the Language Environment run-time library.

run. (1) To cause a program, utility, or other machine function to be performed. (2) A performance of one or more jobs. [I][A] (3) A single, continuous performance of a computer program or routine.

run file. The output of the linkage editor. A program file in a format that is suitable for being loaded into main storage and run. [OSF] See also *module*.

run-time library. A library that is loaded dynamically and used during execution time.

S

S_ISGID bit. Synonym for *set-group-ID mode bit*.

S_ISUID bit. Synonym for *set-user-ID mode bit*.

SAF. System authorization facility.

saved set-group-ID. An attribute of a process that allows some flexibility in the assignment of the effective group ID attribute, when the saved set-user-ID option is implemented. [POSIX.1]

saved set-user-ID. An attribute of a process that allows some flexibility in the assignment of the effective user ID attribute, when the saved set-user-ID option is implemented. [POSIX.1]

scheduler. A computer program designed to perform functions such as scheduling, initiation, and termination of jobs. [A]

scan. (1) To examine sequentially, part by part. [A] See also *browse*. (2) To search records for a specified character string or syntax error.

scope. (1) That part of a source program in which a variable can communicate its value. [OSF] (2) The portion of a program within which a declaration applies. [OSF]

scroll. (1) To move a display image vertically or horizontally to view data that otherwise cannot be observed within the boundaries of the display screen. (2) To move the representation of data vertically or horizontally relative to the terminal screen. There are two types of scrolling: (1) the cursor moves with the data; or (2) the cursor remains stationary while the data moves. [POSIX.2]

search path. The sequence of directories that a command interpreter should search to find the program that the user wants to run. See also *PATH*.

seconds since the Epoch. A value to be interpreted as the number of seconds between a specified time and the Epoch. A Coordinated Universal Time name, specified in terms of seconds (*tm_sec*), minutes (*tm_min*), hours (*tm_hour*), days since January 1 of the year (*tm_yday*), and calendar year minus 1900 (*tm_year*) is related to a time represented as seconds since the Epoch, according to the following expression: If the year < 1970 or the value is negative, the relationship is undefined. If the year ≥ 1970 and the value is nonnegative, the value is related to a Coordinated Universal Time name according to the expression:

$$tm\_sec + tm\_min*60 + tm\_hour*3\ 600 + \\ tm\_yday*86\ 400 + \\ (tm\_year-70)*31\ 536\ 000 + ((tm\_year-69)/4)*86\ 400$$

[POSIX.1]

section. In the vi editor, text that follows a section heading as defined by the *sect=* option. A section begins on lines starting with a formfeed; starting with an open brace; or a macro in the sections variable.

security. (1) The protection of data, system operations, and devices from accidental or intentional ruin, damage, or exposure. [OSF] (2) The protection of computer hardware, software, and data from accidental or malicious access, use, modification, destruction, or disclosure. Tools for the maintenance of security are focused on availability, confidentiality, and integrity. [POSIX.0] (3) Synonym for *data security*.

security administrator. A programmer who manages, protects, and controls access to sensitive information.

segment. (1) A portion of a computer program that may be executed as an entity without the entire computer program being maintained in main storage. [T] (2) A contiguous area of virtual storage allocated to a job or system task. A program segment can be run by itself, even if the whole program is not in main storage. [OSF] (3) A group of display elements.

select. (1) To examine records to find those that satisfy a specified condition. (2) To mark or choose an item in a panel. (3) Contrast with *enter*.

selection. (1) Addressing a terminal or a component on a selective calling circuit. (2) The process by which a computer requests a station to send it a message.

semaphore. (1) An indicator used to control access to a file; for example, in a multiuser application, a flag that prevents simultaneous access to a file. (2) An entity used to control access to system resources. Processes can be locked to a resource with semaphores if the processes follow certain programming conventions.

separator. (1) A punctuation character that separates parts of a command or file. (2) A punctuation character used to delimit character strings.

sequential. Pertaining to the occurrence of events in time sequence with no simultaneity or overlap. [A] Contrast with *concurrent*, *consecutive*.

sequential data set. (1) A data set whose records are organized on the basis of their successive physical positions, such as on magnetic tape. Contrast with *direct data set*. (2) A data set in which the contents are arranged in successive physical order and are stored as an entity. The data set can contain data, text, a program, or part of a program. Contrast with *partitioned data set (PDS)*.

sequential processing. (1) The processing of logical records in the order in which they are accessed. (2) The processing of records in the same order as they exist in a file. Synonymous with *consecutive processing*. (3) Contrast with *random processing*.

serialization. (1) The consecutive ordering of items. (2) In MVS, the process of controlling access to a resource to protect the integrity of the resource.

Serial Line Internet Protocol (SLIP). A transmission line protocol that encapsulates and transfers IP datagrams over asynchronous serial lines. [OSF]

server. (1) A functional unit that provides shared services to workstations over a network; for example, a file server, a print server, a mail server. [T] (2) In a network, a data station that provides facilities to other stations; for example, a file server, a print server, a mail server. [A] (3) On a network, the computer that contains the data or provides the facilities to be accessed by other computers on the network. [OSF] (4) A program that handles protocol, queuing, routing, and other tasks necessary for data transfer between devices in a computer system. [OSF] (5) Contrast with *client*.

ServerPac. A software delivery package consisting of installed products, integrated service, and operational data sets for which IBM has already taken SMP/E installation steps and some post-installation steps. It is one of the two entitled methods for installing z/OS, CBPDO being the other.

server process. In the client-server model of communication, a process that provides services to client processes. [OSF] Contrast with *client process*.

service request block (SRB). A unit of work that is scheduled to execute in another address space.

session. (1) The period of time during which a user of a terminal can communicate with an interactive system—usually, the elapsed time between logon and logoff. (2) The period of time during which programs or devices can communicate with each other. [OSF] (3)

A collection of process groups established for job control purposes. Each process group is a member of a session. Each process is considered to be a member of the session of which its process group is a member. A newly created process joins the session of its creator. A process can alter its session membership. Implementations that support the **setpgid()** function can have multiple process groups in the same session. [POSIX.1] Every process group, and associated process, belongs to a session. Any new process also belongs to the session of the process that created it. (4) In network architecture, an association of facilities that establish, maintain, and release connections for communication between stations. [OSF] (5) In SNA, a logical connection established between two network addressable units (NAUs) that allows them to communicate. For routing purposes each session is identified by the local or network addresses of the session partners. (6) The logical connection between two LUs.

session leader. A process that has created a session. [POSIX.1]

set flags. Flags that can be put into effect with the shell **set** command. [OSF]

setgid bit. Deprecated term for *set-group-ID mode bit*.

set-group-ID mode bit. In setting file access permissions, the bit that sets the effective group ID of the process to the file's group on execution. Synonymous with **S_ISGID** bit.

setuid bit. Deprecated term for *set-user-ID mode bit*.

set-user-ID mode bit. In setting file access permissions, the bit that sets the effective user ID of the process to the file's owner on execution. Synonymous with **S_ISUID** bit.

severity code. A code that indicates how serious an error condition is.

shared address space. Provides the ability for multiple z/OS UNIX processes to run in a single address space.

shared HFS support. Enables systems in a sysplex to share file system data throughout the sysplex; thus, the file systems are available to be accessed (read/write) by all systems participating in shared HFS support.

sh_history. See *history file*.

shared library. A library that contains at least one subroutine that can be used by multiple processes. Programs and subroutines are linked as before, but the code common to different subroutines is combined in one library file that can be loaded at run time and shared by many programs.

shared library object. This is an object that when loaded is placed into the shared library region for system wide sharing.

shared library program. This is a program that when loaded is placed into the shared library region for system wide sharing.

shared library region. This is the area of storage in the system where shared library objects are loaded.

shell. (1) A program that interprets and processes interactive commands from a pseudoterminal or from lines in a shell script. The equivalent in MVS is Time Sharing Option (TSO) and Interactive System Productivity Facility (ISPF). (2) A program that interprets sequences of text input as commands. It may operate on an input stream, or it may interactively prompt and read commands from a terminal. [POSIX.2] Synonymous with command language interpreter. (3) A software interface between a user and the operating system of a computer. Shell programs interpret commands and user interactions on devices such as keyboards, pointing devices and touch-sensitive screens and communicate them to the operating system. (4) The command interpreter that provides a user interface to the operating system and its commands. (5) The program that reads a user's commands and executes them. (6) The shell command language interpreter, a specific instance of a shell. [POSIX.2] (7) A layer, above the kernel, that provides a flexible interface between users and the rest of the system. (8) Software that allows a kernel program to run under different operating system environments. (9) See also *shell program*, *KornShell*.

shell procedure. See *shell script*.

shell program. A program that accepts and interprets commands for the operating system. See also *shell*.

shell prompt. The character string on the command line indicating that the system can accept a command.

shell script. (1) A file of shell commands. If the file is executable, a user can run it by specifying the file's name as a shell command or as an operand on **sh** or on the TSO/E OMVS command. A shell script is like a TSO/E REXX exec. (2) A file containing shell commands. If the file is made executable, it can be executed by specifying its name as a simple command: Execution of a shell script causes a shell to execute the commands within the script. Alternately, a shell can be requested to execute the commands in a shell script by specifying the name of the shell script as the operand to the **sh** utility. [POSIX.2]

shell variables. Facilities of the shell program for assigning variable values to constant names.

signal. (1) A means of informing processes of asynchronous events. (2) A mechanism by which a process may be notified of, or affected by, an event

occurring in the system. Examples of such events include hardware exceptions and specific actions by processes. The term *signal* is also used to refer to the event itself. [POSIX.1] (3) An indication that an asynchronous event completed. A signal is sent to a process. Signals are simulations of *interrupts*. (4) A simple method of communication between two processes. One process can inform the other process when an event occurs. [OSF] (5) A method of interprocess communication that simulates software interrupts.

signal handler. A subroutine called when a signal occurs. [OSF]

signal mask. A mask that defines the set of signals currently blocked from delivery to a process.

simultaneous. Pertaining to the occurrence of two or more events at the same instant of time. [A] Contrast with *concurrent*, *sequential*.

single-precision. (1) The use of one computer word to represent a number, in accordance with the required precision. (2) The specification that causes a floating-point value to be stored in the short format.

single-quote. The character ' —also known as *apostrophe*. [POSIX.2]

slash. (1) The literal character “/”. This character is also known as *solidus* in ISO 8859-1 [B34] [POSIX.1] (2) The character /. UNIX and POSIX-conforming systems use the slash to separate the parts of a filename.

slave. The processor in a multiprocessor environment that operates in user mode. The other processor, the master, one processor is the master and operates in kernel mode, handling the interrupts and system calls. The slave informs the master when making a system call. See also *master*.

SLIP. (1) Serial Line Internet Protocol. (2) Serviceability level indication processing.

SMIT. System Management Interface Tool.

socket. (1) A method of communication between two processes. Sockets allow communication in two directions, in contrast to pipes, which allow communication in only one direction. The processes using a socket can be on the same system or on systems in the same network. (2) A unique host identifier created by the concatenation of a port identifier with a TCP/IP address. (3) An interface, linked with TCP/IP or other protocols, that allows processes on different machines to communicate. Sockets are similar to APPC, but the communication mechanism is transparent: It consists of three layers (socket layer, protocol layer, and device layer). There are two types of sockets: virtual stream sockets and datagram sockets. (4) A port on a specific host; a communication endpoint

that is accessible through a protocol family's addressing mechanism. (5) In TCP/IP, the Internet address of the host computer on which the application runs, and the port number it uses. (6) In interprocess communication, an endpoint of communication. (7) The system call that creates a socket and its associated data structure. (8) A port identifier. (9) Synonym for *port*.

software. (1) Programs, procedures, rules, and any associated documentation pertaining to the operation of a system. [OSF] (2) Programs, procedures, rules, and possibly associated documentation pertaining to the operation of a system. [I][A] (3) The programs, procedures, rules, and any associated documentation pertaining to the operation of a data processing system. [POSIX.0] (4) Contrast with *hardware*.

sort. (1) To rearrange some or all of a group of items, based upon the contents or characteristics of those items. [OSF] (2) To segregate items into groups according to specified criteria. [I][A] Sorting involves ordering, but need not involve sequencing, for the groups may be arranged in an arbitrary sequence. (3) To arrange a set of items according to keys that are used as a basis for determining the sequence of the items, for example, to arrange the records of a personnel file into alphabetical sequence by using the employee names as sort keys. [A]

source. (1) A system, a program within a system, or a device that makes a request to a target. Contrast with *target*. (2) In advanced program-to-program communications, the system or program that starts jobs on another system. (3) Synonym for *source code*. (4) Contrast with *target*.

source code. (1) The input to a compiler or assembler, written in a source language. Synonymous with *source*. Contrast with *object code*. See also *source program*. (2) In the shell command language, input to the command language interpreter; synonym for *shell script*. [POSIX.2] (3) In the C Language Bindings Option, input to a C compiler conforming to the C Standard. [POSIX.2] (4) In another ISO/IEC conforming language, input to a compiler conforming to that ISO/IEC standard. [POSIX.2] (5) The input statements prepared for the following standard utilities: **awk**, **bc**, **ed**, **lex**, **localedef**, **make**, **sed**, and **yacc**. [POSIX.2] (6) A collection of sources meeting any or all of the foregoing meanings. [POSIX.2]

source file. A file that contains source statements for such items as high-level language problems and data description specifications.

source language. A programming language acceptable as input to a translator. [T]

source module. The statements or codes that form input to the assembler. [OSF]

source program. (1) A computer program expressed in a source language. [I][A] See also *source code*. (2)

A set of instructions written in a programming language that must be translated to machine language before the program can be run.

source statement. A statement written in a programming language.

space. (1) A site intended for storage of data, such as a location in a storage medium. (2) A basic unit of area, usually the size of a single character. [A] (3) One or more space characters. [A] (4) The character defined in POSIX.2 2.4 as <space>. The <space> character is a member of the space character class of the current locale, but represents the single character, and not all of the possible members of the class. [POSIX.2]

spawn. To create and start a child process running a named program. The **spawn()** function is the logical combination of fork and exec; its purpose is to avoid the system overhead incurred with fork. The MVS equivalent is multitasking, or attaching.

special character. A character other than a letter or number. For example, *, +, and % are special characters.

special character file. See *character special file*.

special file. A file that provides an interface to input/output devices, a pipe, or a FIFO special file. There is at least one special file for each device connected to the computer. See also *block special file*, *character special file*.

specification. A document that prescribes, in a complete, precise, verifiable manner, the requirements, design, behavior, or characteristics of a system or system component. [POSIX.0]

SRB. Service request block.

stack. (1) An area in storage that stores temporary register information and returns addresses of subroutines. (2) A list constructed and maintained so that the last data element stored is the first data element retrieved. [OSF] (3) In kernel mode, an area that is paged with the user process. The kernel maintains a stack for each process. It saves the process information such as the call chain and local variables used by the kernel for the user process.

standard error (stderr). (1) The place where many programs place error messages: the display screen unless another place is specified with redirection. [OSF] (2) An output stream usually intended to be used for diagnostic messages. [POSIX.2] (3) The conventional name for file descriptor 2. By convention, programs write diagnostics and error messages to this descriptor. Usually, the descriptor refers to the display screen, but it may be changed by redirection. This descriptor is separate from standard output so that error diagnostics are still visible when the output is redirected.

standard input (stdin). (1) The primary source of data going into a command. Standard input comes from the keyboard unless redirection or piping is used, in which case standard input can be from a file or the output from another command. (2) An input stream usually intended to be used for primary data input. [POSIX.2] (3) The conventional name for file descriptor 0. By convention, programs read input from this descriptor. Usually, the descriptor refers to the keyboard, but it may be changed by redirection.

standard output (stdout). (1) The primary destination of data coming from a command. Standard output goes to the display unless redirection or piping is used, in which case standard output can be to a file or another command. (2) An output stream usually intended to be used for primary data output. [POSIX.2] (3) The conventional name for file descriptor 1. By convention, programs write output to this descriptor. Usually, the descriptor refers to the display screen, but may be changed by redirection.

standards. Documents, established by consensus and approved by a recognized body, that provide, for common and repeated use, rules, guidelines, or characteristics for activities or their results, aimed at the achievement of the optimum degree of order in a given context. [POSIX.0]

stanza. A group of lines in a file that together have a common function or define a part of the system. Stanzas are usually separated by blank lines, and each stanza has a name.

started task. In MVS, an address space that runs unattended as the result of a START command. Started tasks are generally used for critical applications. The UNIX equivalent is a daemon.

statement. (1) In a programming language, a meaningful expression that may describe or specify operations and is usually complete in the context of that programming language. (2) An instruction in a program or procedure. (3) In programming languages, a language construct that represents a step in a sequence of actions or a set of declarations. [I] (4) In computer programming, a symbol string or other arrangement of symbols. [A]

static storage. Storage other than dynamic storage. [A]

station. (1) A computer or device that can send or receive data. [OSF] (2) A location on a device at which an operation is performed. (3) An input or output point of a system that uses telecommunication facilities—for example, one or more systems, computers, terminals, devices, and associated programs at a particular location that can send or receive data over a telecommunication line.

status. (1) The current condition or state of a program or device—for example, the status of a printer. [OSF]

(2) The condition of the hardware or software, usually represented in a status code.

stderr. Standard error.

stdin. Standard input.

stdout. Standard output.

STEPLIB. A set of private libraries used to store a new or test version of an application program, such as a new version of a runtime library.

STEPLIB environment. The STEPLIB DD and associated private library allocations that make up a user's MVS program search order environment.

sticky bit. (1) An access permission bit that causes an executable program to remain on the swap area of the disk. Only someone with root authority can set the sticky bit. (2) A permission bit that can be set for a directory or a file. When set for a directory, the sticky bit controls permission to remove or rename a file in the directory. When set for an executable file, the sticky bit allows multiple users to share a single copy of the file.

stop. To end, in a controlled manner, the current processing activity in a computer system because it is impossible or undesirable for the activity to proceed. [OSF] Synonymous with abort.

stopped state. A state in which a device is not available, but still has its device driver loaded and bound in the kernel and is still known by the device driver.

storage. (1) The location of saved information. [OSF] (2) A unit into which recorded text can be entered, retained, and processed, and from which it can be retrieved. [T] (3) A device, or part of a device, that can retain data.

storage administrator. A programmer who controls use of external storage, including the z/OS UNIX file system.

storage device. A functional unit for storing and retrieving data. [OSF]

store. (1) To place data in a storage device. [I][A] (2) To retain data in a storage device. [I][A] (3) Deprecated term for *storage*. (4) Contrast with *retrieve*.

stream. (1) An ordered sequence of characters, as described by the C Standard. [POSIX.2] (2) Sequential input or output from an open file descriptor. [OSF] (3) All data transmitted through a data channel in a single read or write operation. [OSF] Synonymous with data stream. (4) A full-duplex connection between a user process and device or pseudodevice. A stream consists of several linearly connected modules, and is analogous to a shell pipeline, except that data flows in

both directions. The modules communicate almost exclusively by passing messages to their neighbors. [OSF]

stream editor. An editor invoked by the **sed** command, which modifies lines from a specified file, according to an edit script, and writes them to a standard output. [OSF]

string. (1) A linear sequence of entities such as characters or physical elements. Examples of strings are alphabetic string, binary element string, bit string, character string, search string, and symbol string. [OSF] (2) An ordered sequence of bits, octets, or characters, accompanied by the string's length. [OSF] (3) A sequence of characters or numbers. Any sequence of characters, as in *abc*. For the shell, strings should be enclosed by quotes (") to hide any blanks or tabs in the string from the shell.

structure. (1) A variable that contains an ordered group of data objects. Unlike an array, the data objects within a structure can have varied data types. (2) Synonym for *data structure*.

stub. A function that connects with the library but remain outside the library; a protocol extension procedure. Synonymous with *hooking routine*.

subcommand. A request for an operation that is within the scope of work requested by a previously issued command.

subdirectory. In the file system hierarchy, a directory contained within another directory. Directories may be nested to arbitrary depth.

subprogram. A program invoked by another program, such as a subshell. Contrast with *main program*.

subroutine. (1) A sequenced set of instructions or statements that may be used in one or more computer programs and at one or more points in a computer program. [T] (2) A routine that can be part of another routine. [OSF] See also *routine*.

subscript. (1) An integer or variable whose value refers to a particular element in a table or an array. [OSF] (2) A symbol associated with the name of a set to identify a particular subset or element. [I][A]

subshell. An instance of the shell program started from an existing shell program.

substring. A part of a character string.

subsystem. (1) A secondary or subordinate system, usually capable of operating independently of, or asynchronously with, a controlling system. [T] (2) A kernel module or set of modules that extend services beyond the original kernel services.

suffix. A character string attached to the end of a filename that helps identify its file type. [OSF] For example, in */dir/file.c*, the suffix is *c*.

superuser. (1) A system user who operates without restrictions. A superuser has the special rights and privileges needed to perform administrative tasks. The MVS equivalent is a user in privileged, or supervisor, mode. (2) A system user who can pass all z/OS UNIX security checks. A superuser has the special rights and privileges needed to manage processes and files.

superuser authority. The unrestricted ability to access and modify any part of the operating system, usually associated with the user who manages the system.

supervisor. The part of a control program that coordinates the use of resources and maintains the flow of processing unit operations. Synonym for *supervisory routine*.

supervisory routine. A routine, usually part of an operating system, that controls the execution of other routines and regulates the flow of work in a data processing system. [I][A] Synonymous with *executive routine*, *supervisor*.

supplementary group ID. An attribute of a process that is used in determining file access permissions. A process has up to [NGROUPS_MAX] supplementary group IDs in addition to the effective group ID. The supplementary group IDs of a process are set to the supplementary group IDs of the parent process when the process is created. Whether a process's effective group ID is included in or omitted from its list of supplementary group IDs is unspecified. [POSIX.1] In z/OS UNIX, a supplementary group ID is an attribute of a process used in determining file access permissions.

supported. A condition regarding optional functionality. Certain functionality in POSIX is optional, but the interfaces to that functionality are always required. If the functionality is supported, the interfaces work as specified by the particular part of ISO/IEC 9945 (except that they do not return the error condition indicated for the unsupported case). If the functionality is not supported, the interface shall always return the indication specified for this situation. [POSIX.1]

switch. To change processing from the z/OS shell to a TSO/E command mode or to subcommand mode, or to change back to the shell from one of those modes. In TSO/E command mode, you can enter TSO/E commands; in subcommand mode, you can enter OMVS subcommands.

symbol table. A list of symbol names and their associated values, usually in an object or executable file, giving the names of external symbols and their addresses. Such a table is used for linking and debugging.

symbolic link. A type of file system entry that contains the pathname of and acts as a pointer to another file or directory. [OSF]

synchronization. The action of forcing certain points in the execution sequences of two or more asynchronous procedures to coincide in time. [T]

synchronous. (1) Occurring with a regular or predictable time relationship. (2) Pertaining to requests or responses or both that are processed sequentially. (3) Pertaining to transactions that assume control of a terminal or application, such that the requesting user or application cannot resume work until the transaction is complete and control is returned. For example, the keyboard of a terminal user who requests a synchronous transaction remains locked until the transaction completes. (4) Pertaining to two or more processes that depend upon the occurrences of specific events such as common timing signals. [T] (5) Contrast with *asynchronous*.

synchronous transmission. (1) In data communication, a method of transmission in which the sending and receiving of characters is controlled by timing signals. A predetermined number of bits is sent across the line per second, for example, 2400 bits per second. There are no start and stop bits. Contrast with *asynchronous transmission*. (2) Data transmission in which the time of occurrence of each signal representing a bit is related to a fixed time base. [I]

syntax. (1) The rules for the construction of a command, statement, or program. [OSF] (2) The rules governing the structure of a language. [A] (3) The relationship and structure of characters or groups of characters, independent of their meanings or the manner of their interpretation and use. [I][A]

syscall. Synonym for *callable service*.

SYSOUT. A system output stream, or an indicator used in data definition statements, to signify that a data set is to be written on a system output device.

sysplex. A set of MVS systems communicating and cooperating with each other through certain multisystem hardware components and software services to process customer workloads. See also *Parallel Sysplex*.

sysplex couple data set. A couple data set that contains sysplex-wide data about systems, groups, and members that use XCF services. All MVS systems in a sysplex must have connectivity to the sysplex couple data set. See also *couple data set (CDS)*.

Sysplex Timer®. An IBM unit that synchronizes the time-of-day (TOD) clocks in multiple processors or processor slides. External Time Reference (ETR) is the MVS generic name for the IBM Sysplex Timer (9037).

system. (1) The computer and its associated devices and programs. (2) An implementation of POSIX.1.

[POSIX.1] (3) In data processing, a collection of people, machines, and methods organized to accomplish a set of specific functions. [I][A]

system administrator. The person at a computer installation who designs, controls, and manages the use of the computer system.

system call. Synonym for *callable service*.

system dump. (1) A dump of all or part of storage. (2) A copy from storage of all active programs (and their associated data) whenever an error stops the system. [OSF] Contrast with *task dump*.

system management facilities (SMF). An optional control program feature that provides the means for gathering and recording information that can be used to evaluate system usage.

System Management Interface Tool (SMIT). An interactive interface application designed to simplify AIX® system management tasks. SMIT displays a hierarchy of menus that lead to interactive dialogs.

system program. A program that controls some or all of an operating system. Contrast with *application program*.

system programmer. A programmer who plans, generates, maintains, extends, and controls the use of an operating system with the aim of improving overall productivity of an installation.

Systems Application Architecture (SAA). A collection of selected software interfaces, conventions, and protocols that form the framework for developing consistent applications across offerings of the three major IBM computing environments: (a) System/370, (b) System/3x, and (c) Personal Computer. Systems Application Architecture provides consistency in the following areas: (a) programming interface (the languages and services that application developers use in building their software), (b) user access (the design and use of screen panels and user interaction techniques), (c) communications support (the connectivity of systems and programs), and (d) applications (the software built and supplied by IBM and other vendors). Systems Application Architecture results in programming skills that have broad applicability, applications that can be easily transported or that can span systems, and a simple and uniform user access to these applications.

system services. Firmware and software that provide an aggregation of network element functions into a higher level function, and provide an interface to the data contained in the system. [POSIX.0]

System/370. An upward-compatible extension of the IBM System/360™, offering additional function and improved price/performance.

T

table. (1) An array of data each item of which can be unambiguously identified by means of one or more arguments. [I][A] (2) A two-dimensional array in which each item and its position with respect to other items is identified.

tag. A mechanism used to identify certain attributes having some bearing on handling of character data. Some examples are character set identifier, code page identifier, language identifier, country identifier, and encoding scheme identifier.

target. (1) A system, a program within a system, or a device that interprets, rejects, or satisfies, and replies to requests received from a source. [OSF] (2) Pertaining to a storage device to which information is written. (3) Contrast with *source*.

task. (1) A basic unit of work to be accomplished by a computer. The task is usually specified to a control program in a multiprogramming or multiprocessing environment. (2) A basic unit of work to be performed. Some examples include a user task, a server task, and a processor task. [OSF] (3) A process and the procedures that run the process. [OSF] (4) In a multiprogramming or multiprocessing environment, one or more sequences of instructions treated by a control program as an element of work to be accomplished by a computer. [I][A] (5) The basic unit of work for the MVS system.

task dump. A copy from memory of a program that failed along with its associated data. [OSF] Contrast with *system dump*.

TCP/IP. Transmission Control Protocol/Internet Protocol.

tcsh. An enhanced but completely compatible version of the Berkeley UNIX C shell, *csh*.

temporary storage. In computer programming, storage locations reserved for intermediate results. [A] Synonymous with *working storage*.

term. The smallest part of an expression that can be assigned a value.

terminal. (1) A device, usually equipped with a keyboard and a display device, capable of sending and receiving information. (2) A functional unit in a system or communication network at which data may enter or leave. [T] A terminal is a type of workstation. (3) A point in a system or communication network at which data can either enter or leave. [A] (4) A character special file that obeys the specifications of POSIX.1

terminal buffer manager. In Communications Server support, the component that collects input from or output to several terminals in a packet for forwarding.

terminal device. Synonym for *terminal*.

terminal device file. (1) See *character special file*. (2) Synonym for *terminal*.

terminal symbol. A value returned by the lexical analyzer (**yylex**) to represent the smallest elements of the grammar. Typically, a terminal symbol is recognized by the lexical analyzer. [OSF]

terminate. (1) To end. (2) To stop.

testing. The running of a system or a program against a predetermined series of data to arrive at a predictable result for the purpose of establishing the acceptability of the system or program. [T]

text. (1) Human-readable data, containing only printable characters. (2) A type of data consisting of a set of linguistic characters (letters, numbers, and symbols) and formatting controls. (3) In word processing, information for human comprehension that is intended for presentation in a two-dimensional form—for example, information printed on paper or displayed on a screen. Text consists of symbols, phrases, or sentences in natural or artificial languages. [OSF] (4) A graphic representation of information on an output medium. Text can consist of alphanumeric characters and symbols arranged in paragraphs, tables, columns, or other shapes. (5) In ASCII and data communication, a sequence of characters treated as an entity when preceded by one start-of-text and terminated by one end-of-text communication control character. (6) The part of a message that is not the header or control information. (7) The control sections of a load module. (8) Synonym for *program text*.

text file. A file that contains characters organized into one or more lines. The lines do not contain NUL characters and none exceed {LINE_MAX} bytes in length, including the <newline>. Although POSIX.1 does not distinguish between text files and binary files, many utilities produce predictable or meaningful output only when operating on text files. The standard utilities that have such restrictions always specify *text files* in their Standard Input or Input Files subclauses. [POSIX.2] See also *binary file*.

TFS. A temporary, in-memory physical file system that supports in-storage mountable file systems. Normally, a TFS runs in the kernel address space, but it can be run in a logical file system (LFS) colony address space. It is typically mounted when the kernel is started in minimum setup mode. In this environment, the TFS is the in-storage file system and it defaults to the root file system.

thread. (1) A single flow of control within a process. [POSIX.0] (2) A single, sequential flow of control. (3) See also *process*, *task*, *threads*.

threads. A user-level (nonkernel) OSF DCE service, based on the Pthreads interface specified by the POSIX

1003.4a standard (Draft 4), that provides portable facilities that support concurrent programming, allowing an application to perform many actions simultaneously. While one thread executes a remote procedure call, another thread can process user input. The threads service includes operations to create and control multiple threads of execution in a single process and to synchronize access to global data within an application. Because a server process using threads can handle many clients at the same time, the threads service is ideally suited to dealing with multiple clients in client-server-based applications. DCE threads service offer three scheduling policies: first-in-first-out (FIFO), wherein the thread in the highest priority category that has been waiting the longest to run is scheduled first; round-robin (RR), wherein all the threads at the highest priority level are given turns running by timeslicing; and default, wherein each thread is given turns running by timeslicing, with higher-priority threads given longer periods of time to run. The threads service provides three facilities for synchronizing threads within a process: mutual exclusion objects (mutexes), condition variables, and the **join** routine.[OSF]

tilde. The character [POSIX.2]

timeout. (1) An event that occurs at the end of a predetermined time period that began at the occurrence of another specified event. [I] (2) A terminal feature that automatically logs off a user if an entry is not made within a specified period of time. (3) A parameter related to an enforced event designed to occur at the conclusion of a predetermined elapsed time. A timeout condition can be canceled by the receipt of an appropriate timeout cancellation signal. (4) A time interval allotted for certain operations to occur—for example, response to polling or addressing before system operation is interrupted and must be restarted. (5) A request for notification when a certain time of day is reached or a certain interval of time has elapsed; a kernel routine to queue such a request; the interruption of the kernel or a task when a certain time has been reached or has elapsed.

Time Sharing Option (TSO). An operating system option; for the System/370 system, the option provides interactive time sharing from remote terminals.

Time Sharing Option Extensions (TSO/E). (1) The base for all TSO enhancements. It provides MVS users with additional functions, improved usability, and better performance. (2) In the MVS/ESA environment, TSO/E also provides virtual storage constraint relief.

time stamp. The value on an object that is an indication of the system time at some critical point in the history of the object.

token. (1) The smallest independent unit of meaning as defined by either the parser or the lexical analyzer. A token can contain data, a language keyword, an

identifier, or other parts of a language syntax. (2) A string of one or more characters representing a single conceptual unit. For example, the tokens in a programming language include numbers, variable names, keywords, operators, and punctuation characters. (3) In a local area network, the symbol for authority passed among data stations to indicate the station temporarily in control of the transmission medium. [T] (4) A particular message or bit pattern that signifies permission to transmit.

token numbers. Nonnegative integers that represent the names of tokens. [OSF]

TP. Transaction program.

trace. (1) To record a series of events as they occur. (2) To record data that provides a history of events occurring in the system. [OSF] (3) To monitor system performance or aid in debugging programs. (4) The output of such a process. [OSF] (5) A record of the execution of a computer program. It exhibits the sequences in which the instructions were executed. [A]

track. A circular path on the surface of a fixed disk or diskette on which information is magnetically recorded and from which recorded information is read.

transaction. (1) An exchange between a workstation and a program, two workstations, or two programs that accomplish a particular action or result. Some examples are the entry of a customer's deposit and the updating of the customer's balance. [OSF] (2) An exchange between a terminal and another device that accomplishes a particular action or result, for example, recording sales items, processing refunds, recording coupons, handling voids, verifying checks before accepting as tender, and arriving at the amount to be paid by or to a customer. A transaction occurs almost every time a user presses a function key or presses <Enter>. (3) A set of input data that triggers the execution of a specific process or job. (4) A unit of work consisting of an arbitrary number of individual operations all of which will either complete successfully or abort with no effect on the intended resources. A transaction has well-defined boundaries. A transaction starts with a request from the application program and either completes successfully (commits) or has no effect (aborts). Both the commit and abort signify a transaction completion. [POSIX.0] (5) In a batch or remote batch entry, a job or job step.

transactional. Pertaining to an application program that is divided into segments, where each segment typically requests an I/O operation with a terminal user, giving up control to other application program segments for the duration of the I/O operation. See also *conversational*, *interactive*.

transaction application program. (1) An application program executed in response to a transaction request.

Synonymous with transaction program. (2) A program written to meet requirements of a chosen transaction processing (TP) application. Such programs allow a sequence of operations that involve resources such as terminals and databases. The transaction application process specifies transaction boundaries. The transaction application process as defined here is a logical entity and may involve an arbitrary number of processes. [POSIX.0]

transaction program (TP). Synonym for *transaction application program*.

transient. Pertaining to a program or subroutine that does not reside in central storage or to a temporary storage area for such a program.

translation. Synonym for *conversion*.

translation table. Synonym for *conversion table*.

transmission. (1) The sending of data from one place for reception elsewhere. [A] (2) In ASCII and data communication, a series of characters including headings and texts. [A] (3) The dispatching of a signal, message, or other form of intelligence by wire, radio, telegraphy, telephony, facsimile, or other means. [T]

Transmission Control Protocol (TCP). A communications protocol used in Internet and any other network following the U.S. Department of Defense standards for internetwork protocol. TCP provides a reliable host-to-host protocol in packet-switched communication networks and in an interconnected system of such networks. It assumes that the Internet Protocol is the underlying protocol. The protocol that provides a reliable, full-duplex, connection-oriented service for applications.

Transmission Control Protocol/Internet Protocol (TCP/IP). The two fundamental protocols of the Internet protocol suite. The abbreviation TCP/IP is frequently used to refer to this protocol suite. TCP provides for the reliable transfer of data, while IP transmits the data through the network in the form of datagrams. Users can send mail, transfer files across the network, or execute commands on other systems. [OSF] See also *Internet Protocol (IP)*, *Transmission Control Protocol (TCP/IP)*.

trap. (1) A special statement used to catch signals within the z/OS shell.[OSF] (2) An unprogrammed hardware-initiated, conditional jump to a specific address. It is like an interrupt, but triggered by direct action of an executing program, rather than by an external event.

truncate. (1) To remove the beginning or ending elements of a string. (2) To drop data that cannot be printed or displayed in the line width specified or available. (3) To shorten a field or statement to a specified length.

TSO/E. Time Sharing Option Extensions.

TTY. Any device that uses the **termios** standard terminal device interface. TTY devices typically perform input and output on a character-by-character basis.

tuning. The process of adjusting system control variables to make the system divide its resources most efficiently for the workload.

TZ environment variable. An environment variable used by the time service to factor in the application program's time zone offset from GMT.

U

unary operator. An arithmetic operator having only one term. The unary operators that can be used in absolute, relocatable, and arithmetic expressions are: positive (+) and negative (-).

undefined. An indication that POSIX imposes no portability requirements on an application's use of an indeterminate value or its behavior with erroneous program constructs or erroneous data. Implementations (or other standards) may specify the result of using that value or causing that behavior. An application using such behaviors is using extensions. [POSIX.1]

underscore character. A character used in each position of an entry field to indicate its length. This means of indicating entry field length is used on display devices that do not have the underscore attribute. The underscores are replaced by characters typed in the field.

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

unformatted file. A file displayed with the data that is not arranged with particular characters. Contrast with *formatted file*.

UID. See *user ID*.

union. A variable that can hold any one of several data types, but only one data type at a time.

union tag. The identifier that names a union data type.

UNIX. A highly portable operating system originally developed by Bell Laboratories that features multiprogramming in a multiuser environment. UNIX is implemented in the C language. UNIX was originally developed for use on minicomputers but has been adapted on mainframes and microcomputers. It is especially suitable for multiprocessor, graphics, and vector-processing systems. Many of the commands in the z/OS shell are based on similar commands available with UNIX System V.

unmount. To logically disassociate a mountable file system from another file system. The TSO/E command to perform this action is UNMOUNT or UMount.

unpredictable. Pertaining to a situation resulting from a violation of an architecture rule that an implementation is not required to report. Results can include an error report from a threads call, the operating system, or the hardware; a hang or deadlock of the program; or an incorrect operation of the program without indication of error. [OSF]

unspecified. An indication that POSIX imposes no portability requirements on applications for correct program constructs or correct data regarding a value or behavior. Implementations (or other standards) may specify the result of using that value or causing that behavior. An application requiring a specific behavior, rather than tolerating any behavior when using that functionality, is using extensions. [POSIX.1]

update. (1) To modify a master data set with current information according to a specified procedure. (2) An improvement for some part of a system. [OSF]

user. (1) A person or function requiring the services of a computing system. (2) The name associated with an account.

user address space. An address space that has at least one MVS task known to the kernel address space. This address space can contain a shell or an application program that uses z/OS UNIX.

user area. The parts of main storage and disk available to the user.

user ID. (1) A unique string of characters that identifies an operator to the system. This string of characters limits the functions and information the operator can use. (2) A nonnegative integer, which can be contained in an object of type *uid_t*, that is used to identify a system user. When the identity of the user is associated with a process, a user ID value is referred to as a real user ID, an effective user ID, or an (optional) saved set-user-ID. [POSIX.1] (3) The identification associated with a user or job. The two types of user IDs are:

- **TSO/E user ID:** A string of characters that uniquely identifies a TSO/E user or a batch job owner to the security program for the system. The batch job owner is specified on the USER parameter on the JOB statement or inherited from the submitter of the job. This user ID identifies a RACF user profile.
- **z/OS UNIX user ID:** A fullword integer that the security administrator assigns to each MVS user ID. This integer, referred to as the UID, is the sole basis for authority checking against such POSIX-defined resources as hierarchical files.

A user ID is equivalent to an account on a UNIX-type system. (4) A symbol identifying a system user. (5) Synonymous with user identification.

userid. The 8-character name used to represent the user. The userid is specified on login, rlogin, or telnet, and is the basis of z/OS security. The UID is the fullword numeric value that is assigned to a userid and is used for UNIX security checks.

user interface. (1) The means by which a user communicates with a system, program, or device. [OSF] (2) The hardware, software, or both that implements a user interface, allowing the user to interact with and perform operations on a system, program, or device. Examples are a keyboard, mouse, command language, or windowing subsystem.

user mode. In a multiprocessor environment, the slave in a master-slave relationship. One processor operates in kernel mode, the other (slave) in user mode. Kernel mode handles the interrupts and callable service. User mode informs the master when making a callable service. In user mode, a process is carried out in the user's program rather than in the kernel. Contrast with *kernel mode*.

user name. (1) The 1-to-8-character name of the owner of an user address space. The z/OS UNIX user name is:

- For a z/OS UNIX application running in batch, the user ID in the USER parameter on the JOB statement or the TSO/E user ID for an interactive submitter of the job.
- For an interactive application, the z/OS UNIX user name is the TSO/E user ID for the user.

(2) In RACF, one to twenty alphanumeric characters that represent a RACF-defined user. (3) A string that is used to identify a user. [POSIX.1] (4) Synonym for *user ID*.

user option. A choice of appearance or interaction characteristics that programmers give users during the operation of an application.

user profile. (1) A description of a user, including user ID, user name, default group name, password, owner, access authority, and other attributes obtained at login. (2) A file in the user's home directory named **.profile** that contains shell commands that set initial user-defined characteristics and defaults for the session.

utility. (1) A service. In programming, a program that performs a common service function. [OSF] (2) The capability of a system, program, or device to perform the functions for which it is designed. (3) A program that can be called by name from a shell to perform a specific task, or related set of tasks. This program is either an executable file, such as might be produced by a compiler-linker system from computer source code, or a file of shell source code, directly interpreted by the shell. The program may have been produced by the user, provided by the implementer of the POSIX.2 standard, or acquired from an independent distributor. The term *utility* does not apply to the special built-in utilities provided as part of the shell command language. The system may implement certain utilities as shell functions

or built-ins, but only an application that is aware of the command search order described in POSIX.2 3.9.1.1 or of performance characteristics can discern differences between the behavior of such a function or built-in and that of a true executable file. [POSIX.2]

V

valid. True, or conforming to an appropriate standard or authority. [OSF]

value. (1) In programming, the contents of a variable or a storage location. [OSF] (2) A quantity assigned to a constant, variable, parameter, or symbol. See also *argument*.

variable. (1) In programming languages, a language object that can take different values, one at a time. The values of a variable are usually restricted to a certain data type. [I] (2) A quantity that can assume any of a given set of values. [A] (3) A character or group of characters that refers to a value and, in the execution of a program, corresponds to an address. (4) A name used to represent a data item whose value can be changed while the program is running. (5) A name with an associated value. In **awk** and the shell, a variable can be used to store information, remember commands, and be manipulated by commands. (6) See also *array*.

variable-length record. (1) A record having a length independent of the length of other records with which it is logically or physically associated. (2) Pertaining to a file in which the records need not be uniform in length. [A] (3) Contrast with *fixed-length record*.

vector. (1) An array of one dimension. [OSF] (2) A quantity usually characterized by an ordered set of numbers. [I][A]

verify. To determine whether a transcription of data or other operation has been accomplished accurately. [A]

version HFS. Also known as the *root HFS*.

vertical tab. The vertical tab character (<vertical tab>). [POSIX.2]

VFS . Virtual file system.

virtual file system (VFS). A remote file system that has been mounted so that it is accessible to the local user.

virtual host address. In a Communications Server system, an address of the RISC System/6000 associated with the Internet address of an AIX/ESA host.

virtual storage. (1) The storage space that may be regarded as addressable main storage by the user of a computer system in which virtual addresses are mapped into real addresses. The size of virtual storage is limited

by the addressing scheme of the computer system and by the amount of auxiliary storage available, not by the actual number of main storage locations. [I][A] (2) Addressable space that is apparent to the user as the processor storage space, from which the instructions and the data are mapped into the processor storage locations.

virtual storage access method (VSAM). An access method for direct or sequential processing of fixed- and variable-length records on direct access devices. The records in a VSAM data set or file can be organized in logical sequence by a key field (key sequence), in the physical sequence in which they are written on the data set or file (entry sequence), or by relative record number.

Virtual Telecommunications Access Method (VTAM). Deprecated term for *Advanced Communications Function for the Virtual Telecommunications Access Method (ACF/VTAM)*.

visible. A region of a window that is viewable and not occluded on the screen by the user.

vnode . Virtual inode. An object in a file system that represents a file. Unlike an inode, there is no one-to-one correspondence between a vnode and the file system; multiple vnodes can refer to a single file. Vnodes are used to communicate between the upper half of the file system and the file system representations.

volume. (1) A data carrier mounted and demounted as a unit—for example, a reel of magnetic tape or a disk pack. [I] (2) A certain portion of data, together with its data carrier, that can be handled conveniently as a unit. [I] (3) That portion of a single unit of storage that is accessible to a single read/write mechanism—for example, a disk or a magnetic tape reel.

VSAM. Virtual storage access method.

W

wait. (1) A state allowing a parent process to synchronize with the execution of an exit issued by a child process. (2) A state in which no instructions are fetched or executed. Synonymous with *wait state*.

wait state. Synonym for *wait*.

white space. (1) Space characters, tab characters, newline characters, and comments. [OSF] (2) A sequence of one or more spaces or horizontal tab characters. White space is used to separate commands on the command line. (3) A sequence of one or more characters that belong to the **space** character class as defined via the LC_CTYPE category in the current locale. In the POSIX Locale, white space consists of one or more <blank>s (<space>s and <tab>s),

<newline>s, <carriage return>s, <form-feed>s, and <vertical tab>s. [POSIX.2]

wildcard character. Either the * and ? special character that can be used in a file specification to match one or more characters. For example, placing ? in a file specification means any character can be in that position. For another example, the file selector ?a matches the filenames **aa**, **Ba**, **ca**, and so on. The * wildcard character matches any sequence of zero or more characters; thus the file selector a* matches any name beginning with **a**. Synonymous with global character (in an MVS context), pattern-making character. See also *regular expression (RE)*.

window. A division of a screen in which one of several programs being executing concurrently can display information.

word. A contiguous series of 32 bits (four bytes) in storage, addressable as a unit. The address of the first byte of a word is evenly divisible by four. [OSF] Synonymous with fullword.

work area. That portion of central storage that is used by a computer program to hold data temporarily.

working directory. (1) The active directory used to resolve pathnames that do not begin with a slash. In similar systems, a working directory may be called the *current directory* or the *current working directory*. (2) A directory, associated with a process, that is used in pathname resolution for pathnames that do not begin with a slash. [POSIX.1] (3) Synonym for *current directory*. (4) See also *current directory*.

working storage. Synonym for *temporary storage*.

work space. Storage space on a direct access storage device (DASD) used in sorting.

workstation. (1) A device that enables users to transmit information to or receive information from a computer; for example, a display station or printer. [OSF] (2) A functional unit at which a user works. It can be a programmable workstation, such as an IBM PS/2® computer, or a nonprogrammable workstation, such as a terminal. (3) One or more I/O devices from which jobs can be submitted to a host system for processing or to which completed jobs can be returned. (4) A configuration of I/O equipment at which an operator works. [T] (5) A terminal or microcomputer, usually one that is connected to a mainframe or to a network, at which a user can perform applications.

write. To output characters to a file, such as standard output or standard error. Unless otherwise stated, standard output is the default output destination for all uses of the term *write*. [POSIX.2]

write access. In computer security, permission to write to an object. Synonymous with write permission.

write lock. A lock that prevents any other process from setting a read lock or a write lock on any part of the protected area. Contrast with *read lock*. See also *lock*.

write permission. Synonym for *write access*.

X

XCF. Cross-system coupling facility.

Z

z/OS UNIX System Services (z/OS UNIX). z/OS services that support an environment within which operating systems, servers, distributed systems, and workstations share common interfaces. z/OS UNIX supports standard application development across multivendor systems. It is required if you want to create and use applications that conform to the POSIX standard. z/OS UNIX combines the personal power of the workstation, the flexibility of open systems, and the strength of MVS. It supports and fosters a superenvironment of larger operating systems or servers and of distributed systems and workstations that share common interfaces. Users can switch back and forth between the traditional TSO/E interface and the shell interface. UNIX-skilled users can interact with the system, using a familiar set of standard commands and utilities. MVS-skilled users can interact with the system, using familiar TSO/E commands and interactive menus to create and manage hierarchical file system files and to copy data back and forth between MVS data sets and files. Application programmers and users have both sets of interfaces to choose from and, by making appropriate tradeoffs, can choose to mix these interfaces.

Numerics

3270 passthrough mode. A mode that lets a program running from the z/OS shell send and receive a **3270** data stream or issue TSO/E commands.

Index

Special characters

- `__IPC_MEGA` option for `shmat()` 19
- `__MAP_MEGA` option for `mmap()` 19
- `_BPX_ACCT_DATA` environment variable 360
 - customizing 369
- `_BPX_INET_FASTPATH` environment variable
 - customizing 369
- `_BPX_JOBNAME` environment variable 203
 - customizing 85, 369
- `_BPX_SHAREAS` environment variable
 - benefits and side effects of using 342
 - customizing 369
 - improving performance with 342
 - shared address space 342
- `_BPX_SPAWN_SCRIPT` environment variable
 - customizing 369
 - improving performance of shell scripts 342
- `_BPX_USERID` environment variable
 - customizing 369
- `_BPXK_AUTOCVT` environment variable
 - using 246
- `_BPXK_JOBLOG` environment variable 282
 - customizing 369
- `_BPXK_MDUMP` environment variable
 - customizing 369
 - dynamically requesting a `SYSMDUMP` 283
 - used in diagnosing problems 283
- `_BPXK_SETIBMOPT_TRANSPORT` environment variable 356
 - customizing 369
- `_C89_CLIB_PREFIX` variable 208
- `_MAKE_BI` environment variable
 - using 346
- `_map_init` 20
- `_map_service` 20
- `_server_init()` 86
- `/dev` directory
 - explanation of 111
- `/dev/console` 137
- `/dev/fd/n`
 - special character file 137
- `/dev/fdn`
 - special character file 137
- `/dev/null`
 - special character file 136
- `/dev/ptypNNNN`
 - specifying 136
- `/dev/ttypNNNN`
 - specifying 136
- `/etc`
 - file systems allocated by `BPXISHFS` 113
 - installing service into 142
- `/etc` directory
 - customizing configuration files 42, 205
 - migrating 12
 - putting `USERIDALIASTABLE` in 37
- `/etc` file system
 - explanation of 110
 - migrating 12
 - migrating the 12
- `/etc/booksrv/bookread.conf` 207
- `/etc/complete.tcsh`
 - customizing 205
- `/etc/csh.cshrc`
 - customizing 205
- `/etc/csh.login`
 - customizing 204
 - NLS customization
 - for z/OS shell 215
- `/etc/inetd.conf`
 - customizing for `rlogin` 317
- `/etc/init`
 - customizing 198
- `/etc/init.options`
 - customizing 198
- `/etc/log`
 - and `/usr/sbin/init` 198
- `/etc/passwd`
 - explanation of 191
 - in similar open systems 62
- `/etc/profile`
 - customizing 193
 - NLS customization
 - for Japanese 216
 - for z/OS shell 214
 - Simplified Chinese 216
- `/etc/rc`
 - customizing 201
 - sample file 201
- `/samples/init.options`
 - copying to `/etc/init.options` 199
- `/samples/profile`
 - sample of 193
- `/tmp` directory
 - explanation of 111
 - managing the 285
- `/u` directory
 - recommended file system structure 112
- `/usr/sbin`
 - specifying for superusers 197
- `/usr/sbin/ini` 198
- `/usr/sbin/mount` 132
- `/usr/sbin/unmount` 118
- `/var` directory
 - explanation of 111
- `.rhosts` file 317
- `$HOME/.login`
 - customizing 205
- `$HOME/.profile`
 - customizing 197
- `$HOME/.tcshrc`
 - customizing 205
- `$SYSNAME` 161

- \$SYSSYMA
 - mounting file systems using symbolic links 184
- \$SYSSYMR
 - mounting file systems, using symbolic links 184
- \$VERSION 161
- + extended attribute 94

A

- a extended attribute 94
- abend code
 - 0F4 272
 - 422 281
 - EC6 281
- access
 - to directories
 - controlling 89
 - to files
 - controlling 89
 - to z/OS UNIX resources 60
- access ACL 95
 - managing 96
- access control list (ACL)
 - access ACL 95
 - access checks 99
 - auditing 100
 - base ACL entry 95
 - defining default 98
 - directory default ACL 95
 - inheritance 98
 - extended ACL entry 96
 - file default ACL 95
 - inheritance 98
 - inheritance 95
 - managing 96
 - protecting data 93
 - sysplex considerations 100
 - working with 99
- access permission bits
 - setting 90
- accessibility 379
- account number
 - assigning to forked address spaces 359
- accounting information
 - modifying for address spaces
 - BPXOINIT 360
 - OMVS 360
 - MVS 359
- ACEE support
 - limitations of 322
- ACL (access control list)
 - See access control list (ACL)
- address space
 - assigning account numbers for forked 359
 - creating a forked 3
 - dirty 293
 - explanation of 299
 - loading modules from HFS 313
 - ending 251
 - generating jobnames for 364
 - loading programs into 295

- address space (*continued*)
 - making nonswappable 85
 - server 19
- ADDUSER RACF command
 - using with the OMVS segment 62
- ADDUSER RACFcommand
 - defining a user with 61
- administrator
 - in z/OS UNIX 8
- ADSM (ADSTAR Distributed Manager)
 - See Tivoli Storage Manager
- ADSTAR Distributed Storage Manager (ADSM)
 - See Tivoli Storage Manager
- AF_INET sockets
 - BPXPRMxx parmlib member 29
 - processing with common INET (CINET) 353
 - setting up 347
- AF_INET6 sockets
 - BPXPRMxx parmlib member 29
 - setting up 347
- AF_UNIX sockets
 - handling requests for 28
- AFP-authorized extended attributes
 - security implications 66
- allocation
 - file system 113
 - waits 39
- ALLOCxx parmlib member 39
- ALTUSER RACF command
 - altering a user with 61
- AnyNet Sockets over SNA 347
- APAR
 - OW52993 183
- APF authorization for HFS files
 - using sanction lists 298
- APF-authorized extended attribute
 - and sanction lists 298
 - description of 84
- application program
 - interface for z/OS UNIX application 1
 - problem determination 283
- application programmers
 - z/OS UNIX 8
- Application Services
 - customization commands in /etc/rc file 201
 - features of 8
- appropriate privilege
 - determining for
 - daemons 291
- ASCII
 - enhanced
 - file tags 245
 - limitations of 245
 - setting up 246
 - using 245
- ASCII code page 276
- at shell command
 - controlling access to 305
- audit
 - accesses
 - to files 102

- authority checks 66
- AUTHPGMLIST
 - activating sanction lists 105
- AUTOCVT statement 24
 - customizing in BPXPRMxx 35
 - using 246
- AUTOGID keyword
 - defining group identifiers (GIDs) 69
- automatic conversion 245
- AUTOMNT file system type 28
- automount
 - using multilevel security 147
- automount facility
 - automove options supported by MOUNT
 - command 179
 - changing data sets 154
 - keeping automount policy consistent on all systems 176
 - MapName file 149
 - mounting
 - in a sysplex 148
 - NFS data sets 147
 - zFS data sets 147
 - mounting zFS data sets 145
 - naming specific directories 153
 - setting up the 150
 - stopping 154
 - using in a sysplex environment 148
- automount policy
 - displaying 153
- AUTOMOVE parameter in BPXPRMxx 167
- automove system list (SYSLIST)
 - using 164
- AUTOUID keyword
 - assigning UIDs to single users 69

B

- banner page
 - print separator for output 276
- base ACL entry 95
- batch job
 - requesting kernel services 1
- batch shell command
 - controlling access to 305
- BMP (batch message processing)
 - See batch message processing (BMP)
- books
 - installing for OHELP 206
- bookshelf
 - installing for OHELP 206
- BPAM (basic partitioned access method)
 - See basic partitioned access method (BPAM)
- BPAM (basic partitioned access method)
 - access to TFS files 288
 - access to UNIX files 119
 - access to zFS files 146
- BPX messages 282
- BPX_IMAGE_INIT (process image initiation exit) 331
- BPX_PREPROC_INIT (pre-process initiation exit) 331

- BPX_PREPROC_TERM (pre-process termination exit) 331
- BPX.DAEMON 293
 - defining 83
 - setting up for daemons 293
 - setting up security for servers 322
- BPX.DAEMON.HFSCCTL
 - defining 84
 - defining modules to program control 295
 - handling dirty address spaces 299
 - setting up 298
- BPX.DEBUG
 - defining 84
- BPX.FILEATTR
 - defining HFS files as shared library programs 299
- BPX.FILEATTR.APF
 - defining 84
- BPX.FILEATTR.PROGCTL
 - defining 84
 - making programs in HFS files program controlled 297
- BPX.FILEATTR.SHARELIB
 - defining 85
- BPX.JOBNAME
 - defining 85
- BPX.MAINCHECK
 - defining 300
 - setting up for daemons 292
 - setting up for servers 323
- BPX.NEXT.USER
 - defining 84
- BPX.SAFFASTPATH
 - defining 85, 281
- BPX.SERVER
 - defining 85, 315
 - setting up for servers 323
 - setting up security for servers 322
- BPX.SMF
 - defining 85
- BPX.SRV.userid
 - defining 85
- BPX.STOR.SWAP
 - defining 85
- BPX.SUPERUSER
 - defining 86
 - defining superusers 79
- BPX.WLMSEVER
 - defining 86
- BPXAS PROCLIB member
 - used by workload manager (WLM) 3
- BPXBATCH
 - BPXABATSL alias 305
 - starting daemons 305
 - updating for code page support 215
- BPXBATSL entry point 305
- BPXISCDS sample job
 - creating the couple data set 164
- BPXISEC1 sample job
 - explanation of 56
- BPXISSETD REXX exec
 - converting /etc symbolic link to directory 142

- BPXISETS REXX exec
 - converting /etc to symbolic link 142, 162
- BPXISHFS sample job
 - allocating file systems for root and /etc 113
 - role in installation process 11
- BPXISJCL
 - converting /etc in background 142
- BPXISYS1 REXX exec
 - using the 161
- BPXISYS2 REXX exec 162
- BPXISYSR sample job
 - creating sysplex-wide root 161
- BPXISYSS sample job 162
- BPXOINIT address space
 - modifying accounting information for 360
- BPXOINIT started procedure
 - adding to RACF 57
 - cataloged procedure 57
 - CBPDO installation 12
 - ServerPac installation 11
- BPXP006E 284
- BPXPRMLI parmlib member
 - keeping reconfigurable parameters in 261
- BPXPRMXX (sample member) 24
- BPXPRMxx parmlib member
 - changing values without relPLing 27
 - customizing 24
 - CINET 352, 353
 - INET 349
 - customizing for shared HFS 166
 - dynamically adding filetypes to 261
 - dynamically changing values of 259
 - parameters for common INET (CINET)
 - INADDRANYCOUNT 354
 - INADDRANYPORT 354
 - setting user limits 70
 - sharing 23
 - specifying the initial values in IEASYSxx parmlib member 24
 - switching to different members 260
 - syntax checker 24
- BPXTAMD load module 28
- BPXTCAFF 356
- BPXTCINT load module 28
- BPXTUINT load module 28
- BRLM (byte range lock manager)
 - See byte range lock manager (BRLM)
- byte range lock manager (BRLM)
 - centralized 182
 - distributed 182
 - initializing 164
 - setting up distributed 183

C

- C/C++ compiler
 - selecting a previous version 208
 - using the same compiler 208
- c++ utility
 - customizing 207

- c++ utility (*continued*)
 - improving performance of
 - using _MAKE_BI 346
- c89 utility
 - customizing 207
 - improving performance of
 - using _MAKE_BI 346
- caching
 - frequently-read files 345
- CANCEL command
 - ending
 - processes 251
- canonical mode 8
- cataloged procedure
 - BPXOINIT 57
 - daemons 305
 - defining to RACF 89
 - OMVS 42, 57
- CBC.SCBCCMP
 - putting into LPA 335
- CBC.SCCNCMP
 - putting into LPA 335
- CBPDO installation
 - explanation of process 11
 - security requirements for 87
 - setting up BPXOINIT 12
- cc utility
 - customizing 207
 - improving performance of
 - using _MAKE_BI 346
- CDS (couple data set)
 - See couple data set (CDS)
- CEE.SCEERUN 335
- CEEDUMP 283
- character conversion table
 - convert code page 277
 - customizing 278
- character special file
 - creating 135
- CheckSchedEnv() WLM interface 86
- CHECKSUM
 - used to improve TCP/IP performance 6
- Chinese, Simplified
 - customizing
 - for the z/OS shell 213
- CHOWN.UNRESTRICTED 75
 - using 78
- chpriority()
 - enabling 340
- CICS/ESA (Customer Information Control System/ESA) 279
- CINET (common INET) 28
 - binding to a specific socket 355
 - connecting to a specific socket 355
 - customizing BPXPRMxx parmlib member 352
 - displaying network routing information 352
 - setting up for sockets 347
 - specifying parameters in BPXPRMxx parmlib member
 - INADDRANYCOUNT 354
 - starting sockets processing 353

- CINET (common INET) *(continued)*
 - transport affinity 355
 - using specific transports 355
- CINET operand of DISPLAY OMVS 352
- classification rules
 - defining 22
- code page
 - conversion 276
 - customizing 278
 - national 213
- COFVLfxx parmlib member
 - updating 39
- colony address space
 - running a physical file system 43
 - setting up 43
 - setting up security for 59
 - starting outside of JES 43
- commands
 - executing from remote locations, with UUCP 221
- common INET (CINET) 28
 - activating multiple file systems with, for the first time 263
 - binding to a specific socket 355
 - connecting to a specific socket 355
 - customizing BPXPRMxx parmlib member 352
 - displaying network routing information 352
 - setting up for sockets 347
 - specifying parameters in BPXPRMxx parmlib member
 - INADDRANYPORT 354
 - starting sockets processing 353
 - transport affinity 355
 - using specific transports 355
- compiler
 - selecting previous, for Language Environment 208
 - using the same one, for Language Environment 208
- compiler load modules
 - putting into LPA 335
- concatenating
 - libraries to ISPF ddnames 46
 - for the z/OS shell 218
- confighfs shell command
 - expanding the HFS 119
- configuration
 - files
 - TCP/IP 349
 - files, UUCP
 - compiling the 238
 - creating or editing 229
 - how uucico uses 238
 - UUCP (UNIX-to-UNIX copy program) 221
- configuration files
 - customizing 205
- Configuration Wizard 15
- CONNECT RACF command
 - connecting a user to a group with 62
- ConnectWorkMgr() WLM interface 86
- console, system
 - file 137
- ContinueWorkUnit() WLM interface 86
- controlled programs
 - defining modules 295
 - explanation of 299
- conversion
 - code page 276
 - using a character conversion table 277
- CONVERT operand
 - OMVS command
 - converting data with the 278
- COPY DATASET command (DFSMDss)
 - copying a file system 126
- core dump
 - using the 283
- couple data set (CDS)
 - BPXISCDs sample job 164
 - creating the 164
- COUPLExx parmlib
 - defining OMVS CDS to XCF 166
- cover page
 - print separator for output 276
- CPU time limit 190
- CreateWorkUnit() WLM interface 86
- cron daemon 222
 - assigning jobname to 306
 - customizing the 305
 - scheduling UUCP transfers 239
- cron shell command
 - customizing for read-only root HFS 122
- crontab shell command
 - controlling access to 305
- Cross System Coupling Facility (XCF)
 - See XCF (Cross System Coupling Facility)
- CSVDYNEX service
 - defining exits 331
- CTIBPX00 parmlib member 40
- CTIBPX01 parmlib member 40
- CTnBPXxx parmlib member
 - customizing 40
- CTRACE buffer size
 - increasing the 266
- CTRACE statement
 - customizing in BPXPRMxx 30
- curses applications
 - terminfo database 210
- Customer Information Control System/ESA (CICS/ESA)
 - See CICS/ESA (Customer Information Control System/ESA)
- customization
 - /etc/complete.tcsh 205
 - /etc/csh.cshrc 205
 - /etc/csh.login 204
 - /etc/init.options 198
 - /etc/rc 201
 - \$HOME/.login 205
 - \$HOME/.profile 197
 - \$HOME/.tcshrc 205
 - ALLOCxx parmlib member 39
 - BPXPRMxx parmlib member 24
 - c++ 207
 - c89 207
 - carriage conversion tables 278

- customization (*continued*)
 - cc 207
 - CTIBPX00 parmlib member 40
 - CTIBPX01 parmlib member 40
 - CTnBPXxx parmlib member 40
 - daemons
 - cataloged procedure 305
 - IEADMR00 parmlib member 42
 - IEASYSxx parmlib member 24
 - inetd daemon 303
 - ISPF selection panel 46
 - OMVS cataloged procedure 42
 - RACF user profile 192
 - service policies 21
 - Setup Verification Program (SVP) 49
 - shell
 - _C89_CLIB_PREFIX environment variable 208
 - /etc/profile 193
 - electronic mail 211
 - tcsh shell 189
 - electronic mail 212
 - environment variables 191
 - uucpd daemon 303
 - verifying setup choices 49
 - z/OS shell 189, 191
 - environment variables 191
- customizing
 - z/OS shell
 - Japanese 213
 - Simplified Chinese 213

D

- daemon
 - appropriate privileges for 291
 - authority
 - checking 315
 - customizing
 - cron 305
 - inetd 303
 - rlogind 304
 - customizing system for 295
 - defining a service class for 22
 - description of 3
 - IP-supplied 302
 - preparing security program for 294
 - restarting 305
 - security considerations for 291
 - security procedures 308
 - setup problems 310
 - starting 305
 - starting from the shell 305
 - starting in background environment 305
 - sticky bit, checking the 312
 - syslogd
 - starting from the shell 306
 - uucpd
 - customizing 303
- DASD cache
 - performance 333

- data set
 - changing automounted 154
 - protecting 93
- dbx
 - enhanced security checking 301
- DCE
 - recovery 272
- debug
 - APF-authorized programs 84
 - with BPX.SERVER authority 84
- default
 - GIDs 63
 - OMVS segments 63
 - UIDs 63
- defining
 - groups 73
 - z/OS UNIX users 60
- DeleteWorkUnit() WLM interface 86
- Devices file, UUCP 233
- DFS Client
 - colony address space 43
- DFSC file system type 28
- DFSM/MVS 4
- DFSMDss
 - backing up HFS files 128
 - COPY DATASET command 126
- DFSMS
 - managing file systems with 118
- DFSMS/MVS
 - messages 282
- DFSMS/MVS events
 - tracing 265
- DFSMSdfp
 - SMS (System Managed Storage) 16
 - System Managed Storage (SMS) 15
- DFSMSshm
 - backing up HFS files 127
- Dialcodes file, UUCP 234
- Dialers file, UUCP 234
- direct mount 130, 131
- directory
 - auditing accesses to 101
 - controlling access to 89
 - file system 109
- directory default ACL 95
- inheritance 98
- dirty address space
 - defining modules to program control 295
 - explanation of 299
 - loading modules from HFS 313
 - when BPX.DAEMON is defined 293
- dirty environment
 - explanation of 299
 - loading modules from HFS 313
 - when BPX.DAEMON is defined 293
- disability 379
- DisconnectServer() WLM interface 86
- display
 - information about processes
 - ps shell command 266
- DISPLAY command 266

- DISPLAY OMVS command
 - displaying
 - current PFses 261
 - transport providers 352
- Distributed File System (DFS)
 - See DFS (Distributed File System)
- dllload() 18
- documents, licensed xx
- driving system 139
- dump
 - CEEDUMP 283
 - how to take a 268
 - SYSMDUMP 283
- dynamic LPA 334

E

- EBCDIC Latin 1/Open Systems Interconnection code
 - page 1047 276
- EDCRNLIB module 335
- EDCRNLST module 335
- electronic mail 221
 - customizing
 - tcsh shell 212
 - customizing in the shell 211
- EMVSERR return code 293
- Enhanced ASCII
 - file tags 245
 - limitations of 245
 - overview of 245
 - setting up 246
- enhanced program security
 - dbx 301
 - setting up 301
- ENV environment variable file 198
- environment
 - dirty 293
 - explanation of 299
 - loading modules from HFS 313
- environment variable
 - _BPX_ACCT_DATA 360
 - customizing 369
 - _BPX_JOBNAME 203
 - customizing 369
 - defining 85
 - _BPX_SHAREAS
 - customizing 369
 - improving performance with 342
 - shared address space 342
 - _BPX_SPAWN_SCRIPT
 - customizing 369
 - improving performance of shell scripts 342
 - _BPX_USERID
 - customizing 369
 - _BPXK_INET_FASTPATH
 - customizing 369
 - _BPXK_JOBLOG 282
 - customizing 369
 - _BPXK_MDUMP
 - customizing 369
 - dynamically requesting a SYSMDUMP 283
- environment variable *(continued)*
 - _BPXK_MDUMP *(continued)*
 - used in diagnosing problems 283
 - _BPXK_SETIBMOPT_TRANSPORT 356
 - customizing 369
 - _C89_CLIB_PREFIX 208
 - _MAKE_BI 346
 - customizing
 - for shells 191
 - HOME 62
 - LANG 201
 - customizing in /etc/rc 196
 - LOCPATH
 - customizing in /etc/rc 197
 - LOGNAME 62
 - MAIL
 - customizing in /etc/rc 196
 - NLSPATH 201
 - customizing in /etc/rc 196
 - PATH
 - customizing in /etc/rc 195
 - SHELL 62
 - customizing 369
 - STEPLIB
 - customizing 369
 - customizing in /etc/rc 196
 - in /etc/profile 194
 - TMOUT 42
 - TZ 201
 - customizing in /etc/rc 195
- EPHWP00 parmlib member
 - used with man pages 206
- ESCA (extended common service area)
 - See extended common service area (ESCA)
- ESQA (extended system queue area)
 - See extended system queue area (ESQA)
- events
 - tracing 265
- exec sh
 - running the shell command
 - in sample /etc/profile 194
- executable file 4, 109
 - changing owner and group 92
- extended ACL entry 96
- extended attributes
 - + 94
 - 1 94
 - a 94
 - APF-authorized 66
 - using sanction lists 298
 - p 94
 - program control 297
 - s 94
 - shared address space 342
 - shared library 299
 - st_sharelib 337
- extended common service area (ECSA)
 - evaluating virtual storage needs 17
- extended system queue area (ESQA) 17
 - controlling use of 339

external link
accessing MVS load libraries 313

F

F BPXOINIT,DUMP
requesting a SYSMDUMP 283
F BPXOINIT,SHUTDOWN
shutting down z/S UNIX 252
F BPXOINIT,SHUTDOWN=FORKS 254
F BPXONINT,FORCE 254
F OMVS,RESTART 256
F OMVS,RESTART operator command 256
F OMVS,SHUTDOWN 256
explanation of 256
using 256
FACILITY class
BPX.DAEMON
defining 83
setting up for daemons 292, 293
setting up security for servers 322
BPX.DAEMON.HFSCTL
defining 84
defining modules to program control 295
handling dirty address spaces 299
setting up 298
BPX.DEBUG
defining 84
BPX.FILEATTR.APF
defining 84
BPX.FILEATTR.PROGCTL
defining programs in HFS files to program
control 297
setting up 84
BPX.FILEATTR.SHARELIB
defining HFS files as shared library
programs 299
setting up 85
BPX.JOBNAME
defining 85
BPX.MAINCHECK
defining 300
setting up for servers 323
BPX.NEXT.USER
defining 84
BPX.SAFFASTPATH 281
defining 85
BPX.SERVER
defining 85
setting up for servers 323
setting up security for servers 322
BPX.SMF
defining 85
BPX.SRV.userid
defining 85
BPX.STOR.SWAP
defining 85
BPX.SUPERUSER
defining 86
BPX.WLMSEVER
defining 86

FACILITY class profiles
setting up 83
failure
file system 137, 272
file system type 271
kernel 271
recovering from 271
system services 271
fcntl() service
used in file locking 134
FDBX messages 282
field level access
OMVS segment of RACF user profile 68
FIFO special file 135
file
/etc/passwd 62
accessing 91
auditing accesses to 101
caching 345
changing
group 92
owner 92
character special
creating 135
controlling access to 89
description 4, 109
in file system 4, 109
locking
parallel sysplex 182
obtaining security information for 93
permission bits
changing 92
special 135
transferring
UUCP 221
UUCP
Devices 233
Dialcodes 234
Dialers 234
Permissions 234
Systems 229
file default ACL 95
inheritance 98
file descriptor
specifying 137
file descriptor not available message 31
file security packet (FSP)
and RACF 93
definition of 93
file system
/dev
explanation of 111
/etc
explanation of 110
migrating 12
/tmp
explanation of 111
/u
explanation of 112
/var
explanation of 111

- file system (*continued*)
 - copying 126
 - creating 112
 - defining 27
 - failure 137
 - in-memory
 - managing 285
 - managing 109
 - mounting 114
 - mounting, using symbolic links 184
 - organization 343
 - permission bits for 113
 - placement of files 343
 - planned shutdown 252
 - recovering from root problems 137
 - remounting 126
 - root
 - restoring a 137
 - slow response time 119
 - steps for mounting 116
 - sysplex
 - mounting using NFS Client Mount 185
 - moving in a 177
 - unmounting
 - in a non-sysplex environment 119
 - in sysplex 167
- file system clients 186
- file system owner 186
- file system type
 - AUTOMNT 28
 - CINET 28
 - DFSC 28
 - INET 28
 - NFS 28
 - TFS 28
 - UDS 28
 - ZFS 28
- file tag
 - used in Enhanced ASCII 245
- FILE.GROUPOWNER.SETGID 75
 - setting up 90
- FILESYSTYPE statement
 - customizing in BPXPRMxx 27
 - defining CINET 353
 - defining for INET 349
 - dynamically adding 261
 - PARM("")
 - VIRTUAL(max) 28
- FILETAG run-time option
 - using 246
- FOM messages 282
- FOMISCHO sample job
 - using 207
- FOMTLINP module
 - for login 373
- FOMTLOUT module
 - for login 376
- fork() service 19
 - description of 3
- FORKCOPY statement
 - description of 340

- forked address space
 - creating 3
 - defining a service class for 21
- FSP (file security packet)
 - See file security packet (FSP)
- FSUM messages 282
- full function mode
 - explanation of 16
 - switching from minimum mode 16

G

- GFUAINIT load module 28
- GID (group ID)
 - accounting for 359
 - activating supplemental 60
 - assigning 71
 - in an NFS network 71
 - defining 69, 73
 - description 60
 - managing 59
 - setting up default OMVS segments 63
 - upper limits 72
- global resource information
 - displaying 270
- goal mode
 - running in 21
- graphical mode 8
- group
 - changing 92
 - defining 73
 - supplemental 60
- group ID (GID)
 - See GID (group ID)
- group names
 - mapping GID to 69
- group profile, RACF
 - in security 60
- GRPLIST option on the SETROPTS command 60

H

- hangs
 - during initialization 284
- hardware
 - installation 6
- HFS (hierarchical file system)
 - activating for the first time 261
 - APF-controlled extended attribute 84
 - backing up data sets 126
 - BPAM access 119
 - changing mount mode 126
 - copying HFS data sets 126
 - creating user file systems 129
 - data set
 - installing 113
 - locking 134
 - managing 109
 - defining files as APF-authorized
 - using sanction lists 298

- HFS (hierarchical file system) *(continued)*
 - defining files as program-controlled
 - using sanction lists 297
 - defining files as shared library programs 299
 - direct mount 131
 - dirty address space 313
 - dirty environment 313
 - expanding the 119
 - file system program 28
 - handling slow response time 119
 - increasing 118
 - installing products into 142
 - installing service into 140
 - link pack area (LPA) 336
 - loading modules from 313
 - making programs program controlled 297
 - managing 118
 - multisystem enclave support 366
 - multivolume support 129
 - program control attributes 66, 84
 - reducing size of 118
 - security labels 101
 - transporting the 141
- HFS compatibility mode 146
- HFS control
 - setting up 297
- hierarchical file system (HFS)
 - See HFS (hierarchical file system)
- home directory
 - setting up 60, 192
- HOME environment variable
 - value in RACF profile 62

I

- IARVSERV (MVSfunction) 18
- iconv command
 - using to convert data 277
- ICONV TSO/E CLIST
 - using to convert data 277
- IEADMR00 parmlib member 42
- IEASYSxx parmlib member
 - OMVS parameter 15
 - specifying the initial BPXPRMxx parmlib members 24
- IEFIB600 335
- IEFUAVI installation exit 361
- IEFUJI installation exit 361
- IEFUJV installation exit 363
- IEFUSI installation exit 363
- IEWBLINK 335
- IGD messages 282
- IMS/ESA (Information Management System/ESA)
 - See Information Management System/ESA (IMS/ESA)
- in-storage data
 - refreshing 316
- INADDRANYCOUNT
 - in BPXPRMxx parmlib member
 - customizing 354
- INADDRANYPORT
 - in BPXPRMxx parmlib member
 - customizing 354
- INET 28
 - customizing BPXPRMxx parmlib member 349
 - setting up for sockets 347
- inetd daemon 222
 - customizing 303
- Information Management System/ESA (IMS/ESA)
 - batch message processing (BMP) program 279
- initialization
 - diagnosing hangs during 284
- inittab program 307
- inode
 - mounting file systems 114
- installation
 - allocating HFS data set 113
 - COFVLFxx parmlib member, updating the 39
 - hardware 6
 - preparing for RACF 56
 - RACF, preparing for 56
 - security program, preparing the 294
 - security requirements for 87
 - system parmlib members 20
- installation exit
 - BPX_IMAGE_INIT (process image initiation exit) 331
 - BPX_PREPROC_INIT (pre-process initiation) 331
 - BPX_PREPROC_TERM (pre-process termination exit) 331
 - IEFUJI 361
 - IEFUJV 363
 - IEFUSI 363
 - monitoring process activity 331
 - pre-process initiation (BPX_PREPROC_INIT) 331
 - pre-process termination exit (BPX_PREPROC_TERM) 331
 - process image initiation exit (BPX_IMAGE_INIT) 331
- Internal Routing Table 351
- Internet Protocol Version 6 (IPv6)
 - See IPV6
- Interprocess Communication (IPC)
 - managing 272
- IPC (Interprocess Communication)
 - See Interprocess Communication (IPC)
- IPCMSGNIDS statement
 - dynamically changing 260
- IPCS
 - in formatting dumps 282
- IPCSEMNIDS statement
 - dynamically changing 260
- IPCSHMGPPAGES statement
 - dynamically changing 260
- IPCSHMNIDS statement
 - dynamically changing 260
- IPv4
 - setting up 29
- IPv6
 - setting up 29

- IRXISPRM
 - ISPF REXX parameter module
 - customizing 48
- ISPF 6
 - checking the sticky bit 312
 - customizing the menu 46
 - managing zFS file systems 145
 - setting to display Japanese 218
 - Shell 9
 - tasks 9
- ISPF REXX parameter module
 - customizing 48
- ISPF TSO Command Table (ISPTCM)
 - customizing 48
- ISPMMLIB
 - ISPF ddname 46
 - for the z/OS shell 218
- ISPLIB
 - ISPF ddname 46
 - for the z/OS shell 218
- ISPTCM (ISPF TSO Command Table)
 - customizing 48
- ISPTLIB
 - ISPF ddname 46
 - for the z/OS shell 218

J

- Japanese
 - customizing
 - for the z/OS shell 213
 - displaying messages 217
 - issuing messages 218
 - seeing help panels 218
 - setting ISPF for 218
- JCL (job control language)
 - See job control language (JCL)
- JES2
 - relation to z/OS UNIX 278
- JES2 maintenance
 - partial shutdown of z/OS UNIX 254
- JES3
 - relation to z/OS UNIX 279
- job control language (JCL)
 - couple data set format utility 164
- job log
 - writing messages to 369
- job wait timeout (JWT)
 - specifying 42
- jobnames
 - rules used when generating 364
- JoinWorkUnit() WLM interface 86
- JRENVDIRTY reason code 293, 313
- JWT (job wait timeout)
 - See job wait timeout (JWT)

K

- kernel
 - displaying status of 266
 - failure 271

- kernel (*continued*)
 - taking dump of a 268
- kernel services
 - definition of 1
- keyboard 379

L

- l extended attribute 94
- LANG environment variable
 - customizing in /etc/rc 196
 - setting up in /etc/profile 201
- language
 - customizing
 - for the z/OS shell 213
- Language Environment
 - CEEDUMP, preventing a 283
 - run-time library (SCEERUN)
 - putting in LNKLIST 334
 - putting in the LNKLIST 334
 - run-time routines 333
 - SCEERUN
 - using 279
 - SCEERUN2
 - using 279
 - selecting previous compilers 208
 - UNIT=SYSDA 208
 - using STEPLIB to export 343
 - using the same compiler 208
- latch contention
 - locating
 - global resource information 270
- Latin 1 code page 276
- leaf-node connection, UUCP 223
- Library Server 6
- licensed documents xx
- LIMITS parameter value
 - DISPLAY OMVS command 268
- LIMMSG statement
 - customizing in BPXPRMxx 30
- line mode 8
- LINET 348
- link list (LNKLST) 334
- link pack area (LPA)
 - c89 run-time routines 335
 - dynamic 334
 - inserting modules in 333
 - moving HFS executables into the 336
 - putting the run-time library in 334
- list-of-groups checking
 - activating 60
- LISTGRP RACF command
 - obtaining a group profile value with 67
- LISTUSER RACF command
 - obtaining a user profile value with 67
- LNKLST (link list) 334
- load library
 - accessing 313
- load module
 - BPXTAMD 28
 - BPXTCINT 28

- load module *(continued)*
 - BPXTFS 28
 - BPXTUINT 28
 - EZBPFINI 28
 - GFUAINIT 28
 - IOEFSCM 28
- local system
 - UUCP
 - configuring 225
 - creating working directories 238
- lock
 - HFS files 134
- LOCPATH environment variable
 - customizing in /etc/rc 197
- logical mount
 - file systems 112
- login
 - function 373, 376
- LOGNAME environment variable
 - value in RACF profile 62
- logon procedure, TSO/E
 - invoking the shell 189
- LookAt 282
- LookAt message retrieval tool xx
- LPA (link pack area)
 - See link pack area (LPA)
- ls shell command
 - performance 335

M

- mail
 - shell
 - customization 211
 - tcsh shell
 - customization 212
- MAIL environment variable
 - customizing in /etc/rc 196
- mail shell command
 - customizing for read-only root HFS 122
- mail, electronic 221
- make utility
 - improving performance of
 - using _MAKE_BI 346
- man pages
 - enabling the 206
 - setting the path for 195
- managing
 - system limits 338
- MapName file 149
 - reformatting 150
- master file
 - for pseudo-TTY
 - specifying 136
- MAXASSIZE statement
 - changing value 26
 - customizing in BPXPRMxx 31
- MAXCPU TIME statement
 - changing value 26
 - customizing in BPXPRMxx 31
- MAXFILEPROC statement
 - changing value 26
 - customizing in BPXPRMxx 31
- MAXMMAPAREA statement
 - changing value 26
 - customizing in BPXPRMxx 32
- MAXPROCSYS statement
 - customizing in BPXPRMxx 32, 339
 - dynamically changing 260
- MAXPROCUSER statement
 - changing value 26
 - customizing in BPXPRMxx 32, 339
- MAXPTYS statement
 - customizing in BPXPRMxx 33, 339
 - dynamically changing 260
- MAXSHAREPAGES
 - controlling ESQA 18
- MAXSHAREPAGES statement
 - description of 340
- MAXSOCKETS statement
 - customizing in BPXPRMxx 33
 - increasing the value 264
- MAXSPACE
 - determining the value of 270
 - increasing the value of 270
- MAXTHREADS statement
 - changing value 26
 - customizing in BPXPRMxx 34
- MAXTHREADTASKS statement
 - customizing in BPXPRMxx 34
- MAXUIDS statement
 - customizing in BPXPRMxx 34, 339
- MEMLIMIT parameter 32
- mesg shell command 70
 - setting up 207
- message
 - BPXF014D 271
 - BPXP006E 284
- message retrieval tool, LookAt xx
- messages
 - BPX 282
 - dbx 282
 - FDBX 282
 - FOM 282
 - FSUM 282
 - IGD 282
 - writing to job log 369
 - using _BPXK_JOBLOG 282
- migrating
 - /etc directory 12
 - /etc file system 12
- migrating to new releases 11
- minimum mode
 - explanation of 15
 - switching to full function mode 16
- MKDIR keyword
 - defining multiple mount points 29
- mkdir shell command 132
- MKDIR TSO/E command 132
- MKNOD TSO/E command
 - specifying pseudo-TTY files with 136

- mmap() 19
- MMAPAREAMAX
 - specifying 32
- MMAPSIZE parameter 32
- MMS (MVS Message Service)
 - See MVS Message Service (MMS)
- mode
 - compatibility
 - running in 21
 - goal
 - running in 21
- MODIFY command
 - ending processes 250
 - ending threads 251
- module
 - BPXTAMD 28
 - BPXTCINT 28
 - BPXTFS 28
 - BPXTUINT 28
 - EDCRNLIB 335
 - EDCRNLST 335
 - EZBPFINI 28
 - GFSCINIT 28
 - GFUAINIT 28
 - IEFIB600 335
 - IEWBLINK 335
 - IOECMINI 28
 - IOEFSCM 28
 - not defined to program control 313
- monitor
 - processing 329
 - shell processing 329
- mount
 - direct 130
- MOUNT command
 - automove options 179
- mount mode
 - remounting 126
- mount point
 - defining multiple 28, 29
- MOUNT statement
 - customizing in BPXPRMxx
 - defining multiple mount points 28
- mounting
 - file systems 28
 - logical 112
 - root 137
 - HFS data sets 147
 - in a sysplex 148
 - NFS data sets 147
 - user file systems directly 130
 - zFS data sets 145, 147
 - considerations for 146
- msys for Setup
 - customization tasks 51
 - customizing z/OS UNIX 15
 - overview of 51
 - preparing for 52
 - restrictions 52
- multilevel security
 - automount issues 147

- multilevel security *(continued)*
 - HFS file system 101
 - using 101
 - zFS file system 101
- multiple sockets
 - activating for first time 263
- multithreaded server 320
- MVS Message Service (MMS)
 - activating 217

N

- national code page
 - customizing 213
- network
 - connections for z/OS UNIX 7
 - UCP 223
- Network File System (NFS) 6
 - assigning UIDs and GIDs 71
 - BPAM access 119
 - customizing in FILESYSTYPE 28
 - managing files 112
 - mounting data sets 147
- NETWORK statement
 - customizing in BPXPRMxx 29
- NFS (Network File System)
 - See Network File System (NFS)
- NFS Client
 - colony address space 43
 - defining colony address spaces 59
 - mounting in a sysplex 185
 - using supplemental groups for remote communication 60
- NFS file system type 28
- nice()
 - enabling 340
 - prioritizing kernel work 21
- NLSPATH environment variable
 - customizing in /etc/rc 196
 - setting up in /etc/profile 201
- NOAUTOMOVE parameter in BPXPRMxx 167
- non-canonical mode 8
- non-sysplex aware file systems 186
- non-sysplex aware for readonly 186
- Notices 381
- null
 - HFS file
 - specifying 136
- NUUCP user ID
 - for ServerPac and CBPDO installation 226

O

- OBROWSE TSO/E command
 - putting in ISPF menu 46
- OCS (Outboard Communications Server)
 - See Outboard Communications Server (OCS)
- OEDIT TSO/E command
 - putting in ISPF menu 46
- OHELP facility
 - installing books 206

- OHELP TSO/E command
 - displaying online information 206
- OMVS address space
 - modifying accounting information for 360
- OMVS cataloged procedure
 - customizing 42
 - started procedures table 57
- OMVS couple data set (OMVS CDS)
 - identifying to XCF 166
- OMVS parameter
 - IEASYSxx parmlib member 15, 24
 - TRACE command 266
- OMVS segment
 - ADDUSER RACF command 62
 - RACF user profile
 - field level access 68
 - setting up 63
 - setting up default 63
 - TCP/IP user 108
 - verifying 310, 311
- OMVS TSO/E command
 - customization 189
 - customizing code page conversion 278
 - invoking the z/OS shell with 189
 - response time 346
 - specifying Japanese language 218
- OMVSAPPL profile
 - defining the 107
- OMVSDATA subcommand
 - to format problem data 282
- OMVSKERN user ID 57
- one-pack system 142
- OPEN_MAX variable 31
- operation
 - managing 249
- orexecd daemon
 - defining to PROGRAM class 302
- OSHELL
 - updating for code page support 215
- Outboard Communications Server (OCS)
 - See OCS (Outboard Communications Server)
- owner
 - changing 92

P

- p extended attribute 94
- pageable storage
 - evaluating virtual storage needs 17
- Parallel Environment 365
 - WLM multisystem enclave support 366
- parent process
 - ID (PPID)
 - accounting for 359
- PARM parameter
 - options for 287
- parmlib member
 - ALLOCxx 39
 - BPXPRMLI 261
 - BPXPRMxx 24
 - COFVLFxx 39

- parmlib member *(continued)*
 - COUPLExx
 - defining the OMVS CDS to XCF 166
 - CTIBPX00 40
 - CTIBPX01 40
 - CTnBPXxx 40
 - EPHWP00 206
 - IEADMR00 42
 - IEASYSxx
 - OMVS parameter 15
 - SMFPRMxx 4, 42
- partial shutdown
 - for JES2 maintenance 254
- participating group 156
- PATH environment variable
 - customizing in /etc/rc 195
- pax command
 - limitations of 72
- Perform Locked Operation (PLO) instruction
 - used to improve semaphore processing 6
- performance
 - DASD cache 333
 - file system
 - improving 343
 - ideal storage size 333
 - ls shell command 335
 - parmlib limits 338
 - setting duration values 21
 - STEPLIB data sets, using 342
 - z/OS UNIX 333
 - collecting data 329
- permission bits
 - access 90
 - changing 92
 - for file access types 91
 - for newly allocated file systems 113
- Permissions file, UUCP 234
- PGID (process group ID)
 - accounting for 359
- physical file system
 - CINET 28
 - colony address space 59
 - running in a colony address space 43
 - UDS 28
- PID (process ID)
 - using for dump naming 283
- pipe
 - communication between processes 135
- planned shutdown
 - using F OMVS,SHUTDOWN 256
- PLO (Perform Locked Operation) instruction
 - used to improve semaphore processing 6
- PPID (parent process ID)
 - accounting for 359
- pre-process initiation exit (BPX_PREPROC_INIT) 331
- pre-process termination exit
 - (BPX_PREPROC_TERM) 331
- preventive service 139
- print separator
 - output 276
- Print Server 275

- printer
 - designating 275
 - setting up default 275
- printing
 - controlling 275
- prioritizing
 - kernel work 20
- PRIORITYGOAL statement
 - customizing in BPXPRMxx 34, 341
- PRIORITYPG statement
 - customizing in BPXPRMxx 35, 341
- problem determination
 - application program 283
 - daemon setup 310
 - debugger 283
 - server setup 310
 - shell 283
 - taking a dump 268
 - z/OS UNIX 281
- problems
 - re-creating for IBM service 266
- process
 - child 3
 - description 3
 - displaying status of 266
 - ending 250, 251
 - group ID (PGID)
 - accounting for 359
 - parent 3
- process activity
 - monitoring 331
 - tuning 339
- process ID (PID)
 - used for dump naming 283
- process image initiation exit (BPX_IMAGE_INIT) 331
- processing
 - z/OS UNIX
 - managing 275
 - relation to other processing 278
- PROFILE PLANGUAGE setting 218
- program control
 - defining modules to 295
 - dirty address space 299
 - enhanced program security 300
 - finding modules not defined to 313
 - using sanction lists 297
- program control extended attribute
 - in files 66
 - in HFS files 84
 - marking HFS files with the 297
- Program Management Binder 335
- program security
 - setting up 301
- protected resources
 - checking authority for using 322
- protected user ID
 - defining 70
- ps shell command
 - displaying processes with 266
- pseudo-TTY
 - specifying 136

- PTF
 - R609 UW85157 183
 - R703 UW85155 183
 - R705 UW85156 183
- ptrace
 - debugging
 - APF-authorized programs 84
 - programs with BPX.SERVER authority 84
- public UUCP directory 224
- publications
 - on CD-ROM xx
 - softcopy xx

Q

- QuerySchedEnv() WLM interface 86

R

- RACF (Resource Access Control Facility)
 - classes
 - activating 39
 - description of 4
 - establishing 55
 - GIDs, caching 335
 - installing 56
 - managing GIDs and UIDs 59
 - started procedures table
 - BPXOINIT started procedure, adding 57
 - OMVS cataloged procedure, adding 57
 - UIDs, caching 335
 - user profile, OMVS segment
 - field level access 68
 - user, verifying a 60
 - RACF user profile
 - customizing
 - for z/OS shell 192
- raw mode 8
- RDEFINE RACF command
 - defining a field profile with 68
- reason codes 282
 - JRENVDIRTY 293, 313
- recovery
 - DCE components 272
 - file system 272
 - file system type 271
 - system services 271
- remote locations
 - executing commands from, with UUCP 221
- remote system
 - UUCP
 - configuring communication with 228
 - creating working directories 238
- report class
 - defining 21
- RESOLVER_PROC statement 24
- Resource Access Control Facility (RACF)
 - See RACF (Resource Access Control Facility)
- Resource Affinity Scheduling
 - using with Parallel Environment 366

- Resource Measurement Facility (RMF)
 - defining 89
- Resource Measurement Facility (RMF) Monitor III Gatherer
 - defining 89
- Resource Measurement Facility (RMF) 5
- resource name
 - CHOWN.UNRESTRICTED 75
 - FILE.GROUPOWNER.SETGID 75
 - RESTRICTED.FILESYS.ACCESS 75
 - SHARED.IDS 75
 - SUPERUSER.FILESYS 75
 - SUPERUSER.FILESYS.ACLOVERRIDE 75
 - SUPERUSER.FILESYS.CHOWN 75
 - SUPERUSER.FILESYS.MOUNT 75
 - SUPERUSER.FILESYS.PFCTL 75
 - SUPERUSER.FILESYS.QUIESCE 75
 - SUPERUSER.FILESYS.VREGISTER 75
 - SUPERUSER.IPC.RMID 75
 - SUPERUSER.PROCESS.GETPSENT 75
 - SUPERUSER.PROCESS.KILL 75
 - SUPERUSER.PROCESS.PTRACE 75
 - SUPERUSER.SETPRIORITY 75
- resources
 - collecting usage data 329
- response time
 - TSO/E 346
- restarting
 - daemons 305
- RESTRICTED.FILESYS.ACCESS 75
- return code 282
 - EMVSERR 293
 - EMVSPFSFILE 272
 - EMVSPFSPERM 272
- REXX
 - customizing the environment 48
- REXX exec
 - BPXISETS
 - converting /etc to symbolic link 162
 - BPXISYS1 161
 - BPXISYS2 162
- REXX parameter modules
 - ISPF
 - customizing 48
- rlogin
 - command 7
 - problem determination 318
 - setting up for 317
- rlogind daemon
 - customizing the 304
- RMF (Resource Measurement Facility)
 - See Resource Measurement Facility (RMF)
- RMFGAT
 - defining 89
- root directory
 - creating 113
 - in file system 109
- root file system
 - recovery procedure 137
 - restoring a 137

- root HFS
 - definition of 120
- ROOT statement
 - customizing in BPXPRMxx 29
 - defining multiple mount points 29
- RTLS (Run-Time Library Services)
 - See Run-Time Library Services (RTLS)
- run-time library
 - LPA, placing in
 - c89 335
 - managing 279
 - putting in the LNKLIST 334
 - using RTLS 334
 - using STEPLIBs 334
- Run-Time Library Services (RTLS)
 - managing the run-time library with 334

S

- s extended attribute 94
- SAF (system authorization facility)
 - See system authorization facility (SAF)
- sample job
 - BPXISCDs 164
 - BPXISEC1 56
 - BPXISHFS
 - role in installation process 11
 - BPXISYSR 161
 - BPXISYSS 162
 - FOMISCHO
 - using 207
- sanction lists
 - activating 105
 - creating 104
 - formatting rules 103
 - sample 104
 - used by APF-authorized programs 298
 - used by program-controlled programs 297
- SCEELPA data set 333
- SCEERUN
 - putting in the LNKLIST 334
 - used by Language Environment 279
 - using RTLS 334
 - using STEPLIBs LNKLIST 334
- SCEERUN2
 - used by Language Environment 279
- SDSF (System Display and Search Facility)
 - See System Display and Search Facility (SDSF)
- security
 - checking for user authorization to resources 66
 - comparison of UNIX with z/OS UNIX 290
 - considerations
 - for daemons 291
 - for servers 322
 - daemons
 - checklist for 308
 - defining cataloged procedures 89
 - establishing 55
 - improving performance of 346
 - information for files 93
 - level 106

- security (*continued*)
 - Network File System Client (NFSC) 59
 - obtaining security information
 - about users 67
 - for groups 67
 - preparing 56
 - preparing for daemons 294
 - selecting levels for system 108
 - setting up 319
 - setting up for TCP/IP 108
 - threads 320
 - UUCP 223
- security labels
 - automount issues 147
 - HFS file system 101
 - using 100, 101
 - zFS file system 101
- Security Server for z/OS
 - See RACF (Resource Access Control Facility)
- semaphore processing
 - improving performance 6
- SEPHTAB data set
 - used with man pages 206
- server
 - address space 19
 - BPX.SERVER not defined 322
 - checking authority 315
 - processing users without passwords 326
 - setting up 324
 - setting up security level 322
 - setup problems 310
 - using thread-level security 324
 - web
 - SMF records 329
 - WLM server 86
- ServerPac installation
 - explanation of process 11
 - security requirements for 87
- service
 - installing
 - into the HFS 140
 - installing into /etc 142
 - restricting access to 83
 - transporting the HFS 141
- service classes
 - defining for kernel work 21
- service policies
 - customizing 21
- session
 - ID (SID)
 - accounting for 359
- SET OMVS command
 - changing values of BPXPRMxx parmlib members 259
 - for a process 259
 - switching to different BPXPRMxx members dynamically 260
- set-group-ID
 - of executable file
 - creating 92
- set-user-ID
 - of executable file
 - creating 92
- SETOMVS command
 - activating sanction lists 105
 - changing values of BPXPRMxx parmlib members 259
 - for a process 259
 - RESET operand 259
 - SYNTAXCHECK operand 24
- SETOMVS RESET command
 - changing values of BPXPRMxx parmlib members 259
 - dynamically adding physical file systems to BPXPRMxx 261
- SETOMVS SYNTAXCHECK command 27
- SETOMVS SYNTAXCHECK=parmlibmember 29
- setpriority()
 - enabling 340
 - prioritizing kernel work 21
- Setup Verification Program (SVP) 49
- shared address space 94
 - extended attributes 342
- shared HFS
 - availability 185
 - description of 155
 - DFS considerations 188
 - exporting by DFS 188
 - file system clients 186
 - file system owner 186
 - implications during system failure and recovery 178
 - interruptions to file availability 187
 - non-sysplex aware file systems 186
 - non-sysplex aware for readonly 186
 - setting up 161, 186
 - sysplex aware for readonly 186
 - sysplex aware for update file system 186
 - temporary file systems 288
- shared library extended attribute 299, 337
- shared library object 94
- shared library program
 - defining HFS files as 299
- SHARED.IDS 75
 - assigning UIDs to multiple users 70
 - when UID(0) is assigned 83
- shell
 - customizing
 - z/OS 191
 - improving performance of
 - using _BPX_SHAREAS 342
 - initialization of 189
 - interface for z/OS UNIX 1
 - invoking the 189
 - automatically 189
 - with OMVS command 189
- OpenMVS
 - tasks 9
- scripts, improving performance of 342
- setting up 189, 192
- starting daemon from the 305
 - starting in background environment 305

- shell (*continued*)
 - supplying installation-provided shell 190
- SHELL environment variable
 - customizing 369
 - value in RACF profile 62
- shell script
 - using 198
- SHEMMAX parameter 32
- shortcut keys 379
- SHRLIBMAXPAGES
 - controlling ESQA 18
- shutdown
 - partial 254
 - planned 252, 258
 - shared HFS implications 180
- shutting down
 - partial, for JES2 maintenance 254
 - system before an IPL 258
- shutting down z/OS UNIX
 - using F BPXOINIT,SHUTDOWN 252
- SID (session ID)
 - accounting for 359
- SIGDUMP signal 283
- signal
 - SIGDUMP 283
- Simplified Chinese
 - customizing
 - for the z/OS shell 213
- single sockets
 - activating for first time 262
- single stack
 - See INET 349
- single-threaded server 320
- slave file
 - for pseudo-TTY
 - specifying 136
- SMF (system management facilities)
 - See system management facilities (SMF)
- SMFPRMxx parmlib member
 - customizing the 42
 - used in JWT 4
- SMP/E
 - running 80
- SMS (System Managed Storage)
 - See System Managed Storage (SMS)
- SNA (systems network architecture)
 - See systems network architecture (SNA)
- sockets 28
 - activating
 - multiple 263
 - single 262
 - AF_INET 347
 - AF_INET6 347
 - binding to a specific 355
 - BPXPRMxx parmlib statements 28
 - CINET 347
 - connecting through a specific transport 355
 - considerations for a file system 136
 - handling requests 28
 - INET 347
 - sockets 28 (*continued*)
 - processing for
 - common INET (CINET) 353
 - TCP/IP security setup 108
 - UNIX domain 28, 137
- special file
 - file descriptor 137
 - null
 - specifying 136
 - system console 137
 - types 135
 - UNIX domain socket 137
- spool directory, UUCP 242
- SQA (system queue area)
 - See system queue area (SQA)
- st_sharelib extended attribute 337
- start/end exits
 - adding exit routines 332
 - defining 331
- started procedure
 - BPXOINIT
 - CBPDO installation 12
 - ServerPac installation 11
 - kernel 359
- starting
 - daemons 305
- STEPLIB
 - eliminating propagation 343
 - exporting only Language Environment 343
 - preventing excessive searches of 343
 - shell performance 342
 - using to manage the run-time library 334
- STEPLIB environment variable
 - customizing 369
 - customizing in /etc/rc 196
 - in /etc/profile 194
- STEPLIBLIST statement
 - customizing in BPXPRMxx 35
- sticky bit
 - checking 312
 - turning on 341
- storage
 - evaluating virtual 17
- su shell command 291
- SUBFILESYSTYPE statement
 - customizing in BPXPRMxx 29
- superuser
 - assigning attributes 74
 - assigning privileges 77
 - changing from a UID of 0 80
 - defining 74
 - assigning UID(0) 83
 - using BPX.SUPERUSER 79
 - using UNIXPRIV 75
 - setting up \$HOME/.profile 197
 - switching in and out 82
- superuser granularity
 - managing z/OS UNIX privileges 75
- SUPERUSER.FILESYS 75
- SUPERUSER.FILESYS.ACLOVERRIDE 75
- SUPERUSER.FILESYS.CHANGEPERMS 75

- SUPERUSER.FILESYS.CHOWN 75
- SUPERUSER.FILESYS.CHOWN profile 79
- SUPERUSER.FILESYS.MOUNT 75
- SUPERUSER.FILESYS.PFCTL 75
- SUPERUSER.FILESYS.QUIESCE 75
- SUPERUSER.FILESYS.VREGISTER 75
- SUPERUSER.IPC.RMID 75
- SUPERUSER.PROCESS.GETPSENT 75
- SUPERUSER.PROCESS.KILL 75
- SUPERUSER.PROCESS.PTRACE 75
- SUPERUSER.SETPRIORITY 75
- supplemental group
 - activating 60
- SURROGAT class
 - defining servers to process users without passwords 326
- SURROGAT class profile
 - checking the 316
- surrogate profile 324
- SVC dump
 - problem
 - suppressing 282
- SVP (Setup Verification Program)
 - See Setup Verification Program (SVP)
- symbolic link 161
 - command differences 111
 - mounting file systems 184
- syntax checker (BPXPRMxx) 24
- SYS1.KHELP
 - concatenating 46
 - for the z/OS shell 218
- SYS1.LINKLIB system library
 - character conversion tables 277
 - putting program binder in LPA 335
- SYS1.PHELP
 - concatenating
 - for the z/OS shell 218
- SYS1.SAMPLIB
 - sample BPXPRMXX member 24
- SYS1.SBPXEXEC
 - concatenating 46
- SYS1.SBPXMCHS
 - concatenating
 - for the z/OS shell 218
- SYS1.SBPXMENU
 - concatenating 46
- SYS1.SBPXMJPN
 - concatenating 46
 - for the z/OS shell 218
- SYS1.SBPXPCHS
 - concatenating
 - for the z/OS shell 218
- SYS1.SBPXPENU
 - concatenating 46
- SYS1.SBPXPJPN
 - concatenating 46
 - for the z/OS shell 218
- SYS1.SBPXTCHS
 - concatenating
 - for the z/OS shell 218
- SYS1.SBPXTENU
 - concatenating 46
- SYS1.SBPXTJPN
 - concatenating 46
 - for the z/OS shell 218
- SYSCALL host command environment 48
- SYSEXEC
 - ISPF ddname 46
- SYSHELP
 - ISPF ddname 46
 - for the z/OS shell 218
- SYSLIST
 - See automove system list (SYSLIST)
- syslogd daemon
 - starting from the shell 306
- SYSMDUMP 283
 - dynamically requesting a 283
 - preventing a CEEDUMP 283
 - specifying 42
 - specifying where it is to be written to 369
- SYSOMVS parameter value
 - DISPLAY TRACE command 268
 - TRACE command 265
- sysout (system output data set)
 - print separator for output 276
- sysplex
 - access control list (ACL) 100
 - automount facility 148
 - automount policy 176
 - BPXISYS1 REXX exec 161
 - BPXISYSR sample job 161
 - byte range lock manager (BRLM) 182
 - character-special files 135
 - couple data set (CDS) 164
 - cross system coupling facility (XCF) 164
 - customizing BPXPRMxx for shared HFS 166
 - DFS considerations 188
 - exporting by DFS 188
 - FIFO special files 135
 - file lock 182
 - HFS sharing support 162
 - moving file systems in a sysplex 177
 - NFS client mounts 185
 - setting up distributed BRLM 183
 - signaling services 187
 - symbolic links 162
 - sysplex root 161
 - UNIX domain socket address file 135
 - unmounting file system 167
 - version HFS
 - mounting read-only 162
- sysplex aware for readonly 186
- sysplex aware for update file system 186
- sysplex root
 - creating the 161
- SYSPROC
 - ISPF ddname 46
- SYSROOT 15
- system administrator
 - in z/OS UNIX 8

- system authorization facility (SAF)
 - enabling fastpath support for 281
 - z/OS UNIX services 93
- system completion code 281
- system console file
 - /dev/console 137
 - specifying 137
- System Display and Search Facility (SDSF) 5
- system HFS 162
- system limits
 - defining 30
 - displaying status 268
 - managing 338
- System Managed Storage (SMS)
 - for HFS data 109
 - used in full function mode 16
 - used in minimum mode 15
- system management facilities (SMF) 4
 - managing accounting for UNIX workloads 359
 - obtaining data 329
 - Record Type 30 329
 - Record Type 34
 - preventing 330
 - Record Type 35
 - preventing 330
 - Record Type 74 330
 - Record Type 80 330
 - Record Type 92 330
 - user application support 329
 - web server 329
- system output data set (sysout)
 - print separator for output 276
- system programmer
 - in z/OS UNIX 8
- system queue area (SQA) 17
- system-shared library programs 337
- Systems file, UUCP 229
- systems network architecture (SNA) 6

T

- TAG statement 24
- talk shell command 70
 - setting up 207
- tar command
 - limitations of 72
- target system 139
- task
 - using ISPF shell 9
- tasks
 - activating a single sockets file system for the first time
 - steps for 262
 - activating multiple file systems for the first time with Common INET
 - steps for 263
 - activating MVS Message Service (MMS)
 - steps for 217
 - activating sanction lists
 - steps for 105

tasks *(continued)*

- activating the HFS file system for the first time
 - steps for 261
- adding another sockets file system to an existing Common INET configuration
 - steps for 264
- adding the syscall command to the default environment
 - steps for 48
- assigning selected users to transfer file ownership
 - steps for 77
- assigning UIDs and GIDs
 - steps for 80
- checking OMVS security information about a group
 - steps for 67
- checking server authority
 - steps for 315
- checking the daemon authority
 - steps for 315
- checking the SURROGAT class profile
 - steps for 316
- creating a cataloged procedure for a TFS
 - steps for 45
- creating sanction lists
 - steps for 104
- customizing BPXPRMxx for CINET
 - steps for 353
- customizing for your national code page
 - roadmap 213
- customizing the login file for the tcsh shell
 - steps for 216
- customizing the login file for the z/OS shell
 - steps for 216
- customizing the mail utility
 - steps for 125
- customizing the rlogind daemon
 - steps for 304
- customizing the system for IP-supplied daemons
 - steps for 302
- customizing the uucpd daemon
 - steps for 303
- defining HFS files as shared library programs
 - steps for 299
- defining programs from load libraries to program control
 - steps for 296
- defining RACF groups as z/OS UNIX groups
 - steps for 73
- defining terminals or workstations for a terminfo database
 - steps for 210
- displaying messages in Japanese
 - steps for 217
- ending processes
 - steps for 250
- finding modules that were not defined to program control
 - steps for 313
- increasing the MAXSOCKETS value
 - steps for 263, 264

tasks (continued)

- keeping automount policy consistent
 - steps for 177
- maintaining the security level of the system
 - steps for 107
- making programs in HFS files program controlled
 - steps for 297
- managing operations
 - overview 249
- mounting file systems
 - steps for 116
- mounting the temporary file system
 - roadmap 285
 - steps for 286
- obtaining security information about users
 - steps for 67
- preparing the security program for daemons
 - steps for 294
- recovering from file system problems with the root
 - steps for 138
- setting the APF-authorized attribute in HFS files
 - steps for 298
- setting up BPX.SUPERUSER
 - steps for 79
- setting up default OMVS segments
 - steps for 63
- setting up distributed BRLM
 - steps for 183
- setting up Enhanced ASCII
 - roadmap 245
 - steps for 246
- setting up enhanced program security
 - steps for 300
- setting up field-level access
 - steps for 68
- setting up for rlogin
 - steps for 317
- setting up for security
 - roadmap 55
- setting up for your national code page
 - steps for 213
- setting up HFS Control
 - steps for 297
- setting up security procedures for daemons
 - steps for 308
- setting up shared HFS in a sysplex
 - roadmap 155
- setting up the automount facility
 - roadmap 147
 - steps for 150
- setting up the CHOWN.UNRESTRICTED profile
 - steps for 78
- setting up the FILE.GROUPOWNER.SETGID profile.
 - steps for 90
- setting up the inetd daemon
 - steps for 303
- shared HFS in a sysplex
 - overview 155
- shutting down for JES2 maintenance
 - steps for 254

tasks (continued)

- shutting down z/OS UNIX
 - steps for 257
- shutting down z/OS UNIX using F OMVS,SHUTDOWN
 - overview 256
- shutting down z/OS UNIX, using F BPXOINIT,SHUTDOWN
 - steps for 252
- using external links to access MVS load libraries
 - steps for 313
- using zFS
 - overview 145
- verifying the group OMVS segment
 - steps for 311
- verifying the sticky bit
 - steps for 312
- verifying the user OMVS segment
 - steps for 310

TCP/IP

- /etc/resolv.conf 358
- AF_INET sockets 29
- AF_INET6 29
- configuration files 349
- customizing BPXPRMxx 30
- description of 5
- improving performance with CHECKSUM 6
- introduction to 5
- protocol information 358
- resolver file 358
- security setup 108
- service information 357
- socket considerations
 - security setup 108
- started task user ID 108
- user, security setup for 108

tcsh shell

- setting up 189

telnet daemon 3

temporary file system (TFS)

- BPAM access 288
- cataloged procedure 45
- file system type 28
- handling requests for the 28
- managing 285
- mounting the 286
- running in a colony address space 45
- shared HFS environment 288
- SYSROOT 15
- unmounting 287

term

- z/OS UNIX and z/OS equivalents 8

terminal character special file

- specifying 136

terminal connection, UUCP 223

terminal definitions

- terminfo database 210

terminal group name

- defining 70

terminfo database

- customizing 210

- TFS (temporary file system)
 - See temporary file system (TFS)
- TGET WAIT 346
- thread
 - customizing the RACF identity of 320
 - ending 251
 - setting up 319
- tic utility
 - customizing 210
- time limit
 - CPU
 - determining the 190
- Tivoli Storage Manager
 - backing up HFS files 128
- TMOUT environment variable
 - used in job wait timeout 42
- TRACE command 265
- trace data
 - filtering 266
- tracing
 - events 265
 - using the CTnBPXxx parmlib members 40
- tracing events
 - in DFSMS/MVS 265
 - in z/OS UNIX 265
- transfer files between systems with UUCP 221
- Transport Affinity 355
- transport driver
 - default 352
- transports
 - displaying network routing information 352
 - requesting affinity 355
 - using for common INET (CINET) 355
- TSO/E (Time Sharing Option Extensions) 5
 - Japanese messages 218
 - seeing translated help panels 218
 - seeing translated messages on terminals 218
- TTY group name
 - CS remote terminals 70
 - talk and write commands 70
- tuning
 - parmlib limits 338
 - process activity 339
 - processing 329, 333
- TZ environment variable
 - customizing in /etc/rc 195
 - setting up in /etc/profile 201

U

- UDS file system type 28
- UID (user ID)
 - assigning 71
 - in an NFS network 71
 - to multiple users 69
 - assigning to single users 69
 - changing from UID(0) to a nonzero UID 80
 - defining
 - to RACF 60
 - defining protected 70
 - description 60

- UID (user ID) *(continued)*
 - managing 59
 - mapping UID to 70
 - setting up default OMVS segments 63
 - upper limits 72
- UNIT=SYSDA
 - using a system that doesn't have it 208
- UNIX domain socket 28, 137
- UNIX domain socket address file
 - socket considerations
 - for a file system 136
- UNIX System Services
 - publications
 - on CD-ROM xx
 - softcopy xx
- UNIX-to-UNIX copy program (UUCP)
 - See UUCP
- UNIXPRIV class
 - managing z/OS UNIX privileges 75
- UNMOUNT parameter in BPXPRMxx 167
- unmounting
 - in sysplex 167
 - unmounting 28
- user authority 74
- user ID (UID)
 - See UID (user ID)
- user limits
 - establishing 70
- user OMVS segment
 - verifying 310, 311
- user processes
 - taking dump of a 268
- user profile, RACF
 - customizing
 - for z/OS shell 192
 - in security 60
- user-shared library programs 337
- USERIDALIASTABLE statement
 - customizing in BPXPRMxx 37
- users
 - RACF verification 60
 - z/OS UNIX
 - defining 60
- uucc shell command 222
- uucico daemon 222
- uucico shell command
 - configuration files, using 238
- UUCP
 - chat script 231
 - escape characters 233
 - commands 222
 - configuration 221
 - configuration files
 - compiling the 238
 - creating or editing 229
 - controlling calls to each system 240
 - cron transfers 239
 - daemons 222
 - Devices file 233
 - Dialcodes file 234
 - Dialers file 234

- UUCP (*continued*)
 - directories 222
 - execute permissions 225
 - files 222
 - leaf-node connection 223
 - local system
 - configuring the 225
 - creating working directories 238
 - lock files 243
 - log files 243
 - maintenance 242
 - network 223
 - password changes, notifying remote systems 243
 - Permissions file 234
 - public directory 224
 - recorded events, displaying 243
 - remote systems
 - configuring 228
 - creating working directories 238
 - security 223
 - setting up 207
 - spool directory 242
 - status files 243
 - Systems file 229
 - terminal connection 223
 - testing the connection 241
 - working files 243
- uucp shell command 222
 - customizing for read-only root HFS 122
- uucpd daemon 222
 - customizing the 303
- uulog shell command 222, 243
- uname shell command 222
- uupick shell command 222
- uustat shell command 222
- uuto shell command 222
- uux shell command 222
- uuxqt daemon 222

V

- vendor-written programs
 - giving daemon authority to 309
- version HFS
 - mounting read-only 162
- virtual lookaside facility (VLF)
 - caching UIDs and GIDs 335
 - updating COFVLFxx 39
- VIRTUAL(max)
 - storage use 28
 - virtual storage 28
- VLF (virtual lookaside facility)
 - See virtual lookaside facility (VLF)

W

- waits
 - allocation 39

- WLM (workload manager)
 - See workload manager (WLM)
- workload manager (WLM)
 - BPXAS PROCLIB member 3
 - controlling access to server functions 86
 - facility class profile 86
 - interfaces 86
 - multisystem enclave support for Parallel Environment 366
 - prioritizing kernel work 20
 - Resource Affinity Scheduling for Parallel Environment 366
 - response time goals 346
 - running in goal mode 21
 - service policies 21
- write shell command 70
 - setting up 207

X

- XCF (Cross System Coupling Facility)
 - initializing 164

Z

- z/OS Managed System Infrastructure for Setup
 - See msys for Setup
- z/OS shell
 - customizing 191
 - setting up 189
- z/OS UNIX Configuration Wizard 15
- z/OS UNIX customization task
 - description of 51
- z/OS UNIX System Services Application Services
 - customization commands in /etc/rc file 201
 - features of 8
- zFS (zFS(File System)
 - introduction 6
- zFS (zSeries File System)
 - automount facility 147
 - BPAM access 146
 - features of 145
 - FILESYSTYPE statement 28
 - HFS compatibility mode 146
 - ISPF shell 145
 - mounting considerations 146
 - mounting data sets 145
 - multilevel security 101
 - security labels 101
 - using 145
- zFS (zSeries File System)
 - shared sysplex considerations 145
- zFS aggregates
 - remounting 126
- ZFS file system type 28
- zSeries File System
 - See zFS (zSeries File System)

Readers' Comments — We'd Like to Hear from You

z/OS
UNIX System Services Planning

Publication No. GA22-7800-05

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name

Address

Company or Organization

Phone No.



Cut or Fold
Along Line

Fold and Tape

Please do not staple

Fold and Tape



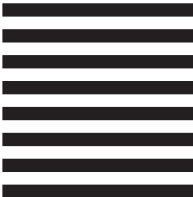
NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Department 55JA, Mail Station P384
2455 South Road
Poughkeepsie, NY
12601-5400



Fold and Tape

Please do not staple

Fold and Tape

Cut or Fold
Along Line



Program Number: 5694-A01, 5655-G52

Printed in USA

GA22-7800-05

