z/OS

UNIX System Services
APAR OA14285

z/OS

UNIX System Services
APAR OA14285

About this document

This document supports APAR OA14285 for z/OS® UNIX System Services. This document is only available on the z/OS UNIX Web site at:
http://www.ibm.com/servers/eserver/zseries/zos/unix/release/apar.html

© Copyright IBM Corp. 1996, 2005

Contents

© Copyright IBM Corp. 1996, 2005

Chapter 1. Changes for z/OS UNIX System Services Planning

Establishing UNIX security

Overview of establishing UNIX security

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. For more information on multilevel security refer to *z/OS Planning for Multilevel Security and the Common Criteria*.

Use the following as reference documents:
- *z/OS Security Server RACF 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 14 "Steps for setting up field-level access for the OMVS segment of a user profile" on page 14
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 24 "Steps for setting up the CHOWN.UNRESTRICTED profile" on page 25
Assigning superuser authority to users	"Steps for setting up BPX.SUPERUSER" on page 26
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 27
Defining RACF groups to z/OS UNIX groups	"Steps for creating z/OS UNIX groups" on page 19

© Copyright IBM Corp. 1996, 2005

Subtasks	Associated procedure (see . . .)
Setting up the FILE.GROUPOWNER.SETGID profile	"Steps for setting up the FILE.GROUPOWNER.SETGID profile" on page 39
Setting up sanction list processing	"Steps for creating a sanction list" on page 52 "Steps for activating the sanction list" on page 53
Maintaining the security level of the system	"Steps for maintaining the security level of the system" on page 55

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 in the chapter "Setting up for daemons:"
- "Comparing UNIX security and z/OS UNIX security"
- "Establishing the correct level of security for daemons"

Support for mixed-case passwords

When the installed security product supports mixed-case passwords, the BPX1PWD, BPX1SEC, and BPX1TLS callable services will also support mixed-case passwords.

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, "Setting up for daemons," for rlogin security considerations
- Chapter 18, "Preparing security for servers," for information about preparing security for servers
- "Steps for preparing the security program for daemons" in chapter 17

Steps for preparing RACF

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

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

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

   Example:

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

- When you create the RACF user ID for OMVSKERN, use the NOPASSWORD option to create it as a protected user ID. Protected user IDs 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 OMVSGRP to any MVS resources, unless programs you start using **/etc/rc** need to be permitted to these resources. For more information, see the section on customizing **/etc/rc** in the *Customizing the shells and utilities* chapter.

  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:

    ```
    HOME('/')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 name that the OMVS address space will be assigned.
   - You must decide whether to mark OMVS (the kernel) trusted for access. Making the kernel trusted is useful for giving the kernel access to any local data set that it wants to mount. If you do not mark the kernel trusted for local access, set up profiles so that the kernel user ID has access to any local data set that it needs to mount. For information about trusted attributes, see

the section on associating started procedures with user IDs in *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 on page 5 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.

     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.

     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:

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

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

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

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

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 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.

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

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

   The home directory should be fully qualified ('**/u/john**'). If a home directory is partially specified (for example, **john**) problems 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.

   Example: 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.

   ```
   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.

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

     To use z/OS UNIX, 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') WAACCNT(12345678)
        WABLDG(507_PARK_PLACE) WAROOM(124)
        WADEPT(ENGNG555) WAADDR1(WIDGET_INC) WAADDR2(NEW_YORK)
        WAADDR3(NEW_YORK) WAADDR4(10002))
```

The DFLTGRP parameter places user ID JOHN in the RACF group ENGNGP7, which has an OMVS GID of 678 (see "Creating z/OS UNIX groups" on page 19). 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 for those users. Then the dub will fail when those users attempt to use a UNIX service. If users must be dubbed (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.

In general, users running with the default OMVS segment are still using system resources just like any other user. In particular, the MAXPROCUSER value is also affected by the default OMVS segment. This value is a measure of the number of processes on the system that any single user (identified by UID) can have running at the same time.) It is incremented for default OMVS segment users when a user process is spawned or forked. However, the MAXPROCUSER value is not incremented when a default OMVS SEGMENT user is “blind-dubbed”, as happens when FTP sessions are logged on or created. Resources are still consumed by these.

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**. You have 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')
  ```

  Any attempts to enter the shell are terminated.

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.

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.

   You do not need to set the UACC or define an access list for BPX.DEFAULT.USER because they will be ignored.

   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.

Guideline: 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 suggested 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 the section about USERIDALIASTABLE in the "Customizing z/OS UNIX" chapter.

Security implications

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

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

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 30.

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

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

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

1. You can link-edit the program into an APF-authorized library and turn on the sticky bit, using the **chmod** command. (For an explanation about 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 7). 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 path name for their home directory. Set these permissions for each directory using the **chmod** command and either the MODE operand of the TSO/E MKDIR command or the *mode* option of the **mkdir** command that creates a directory. For more information, see "Controlling access to files and directories" on page 38.

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 ENGNGP7 OMVS NORACF
   ```

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

Obtaining security information about users

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: **Before you begin:** You need to know which users need to have field-level access.

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

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

   Example:

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

---

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

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. Users with UPDATE access can become a superuser by changing the UID to 0.

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

---

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

   Example:

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

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

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

Defining group identifiers (GIDs)

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

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

Guideline: Do not assign the same GIDto multiple RACF groups. If you do, control at an individual group level is lost because the GID is used in z/OS UNIX security checks. RACF groups that have the same GID assignment are treated as a single group during z/OS UNIX security checks. They must use the SHARED keyword of the RACF ADDGROUP or ALTGROUP command if the SHARED.IDS profile is defined in the UNIXPRIV class. For more information about SHARED.IDS, see *z/OS Security Server RACF Administrator's Guide*.

If you are using NFS, see “Assigning UIDs and GIDs in an NFS network” on page 17 for more information.

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

Defining user identifiers (UIDs)

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

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

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 17 for more information.

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

Setting limits for users: You can control the amount of resources consumed by certain z/OS UNIX users by setting individual limits for these users. The resource limits for the majority of z/OS UNIX users are specified in the BPXPRMxx parmlib member. Instead of assigning superuser authority to application servers and other users so they can exceed BPXPRMxx limits, you can individually set higher limits to these users, as discussed in "Handling process limits in z/OS UNIX" on page 73. Setting user limits allows you to minimize the number of assignments of superuser authority at your installation and reduces your security risk.

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

ASSIZEMAX	Maximum address space size (`RLIMIT_AS`)
CPUTIMEMAX	Maximum CPU time (`RLIMIT_CPU`)
FILEPROCMAX	Maximum number of files per process
MEMLIMIT	Maximum size of storage above the bar
MMAPAREAMAX	Maximum memory map size
PROCUSERMAX	Maximum number of processes for this UID
THREADSMAX	Maximum number of threads per process

After you set individual user limits for users who require higher resource limits, you should consider removing their superuser authority, if they have any. You should also reevaluate your installation's BPXPRM*xx* limits and consider reducing these limits. See the section on customizing BPXPRMxx parmlib members in the chapter about customizing z/OS UNIX 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 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 34.

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

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

Managing user and group assignments

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

- Use AUTOGID or AUTOUID to have UIDs and GIDs automatically assigned to the user. These are keywords on RACF commands that you use to define users and groups. This is the suggested method.
- Use the 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 can access any of user SMORG's files because they both have UID(7).

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

- The users share the same MAXPROCUSER limit, which is defined in the BPXPRMxx 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 the SHARED.IDS profile is defined in the UNIXPRIV class.

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

- The groups share ownership and access to the same files.
- Security audit records show the GID, but do not show the RACF group if it was in the supplemental group list; see "Activating supplemental groups" on page 6.
- 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 the SHARED.IDS profile is defined in the UNIXPRIV class.

Upper limits for GIDs and UIDs

RACF allows for UIDs and GIDs within the range of 0–2 147 483 647. However, the **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 and 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 and 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 or GID, if the USTAR format was used during dumping, **pax** and **tar** will first attempt to set the UID or GID using the user/group name stored in the archive. (Of course, the user must have the appropriate privileges to set the UID or GID). If this name is defined on the target system, then the UID or GID is set to whatever UID or GID is associated with the name defined on the target system. (The UID or GID is set, whether or not it matches the UID or GID in the archive, which means that this could be a problem if the name stored on the target system is coincidental rather than intentional). If the name is not defined on the system, or if the archive is using the original tar format, then the UID or GID stored in the archive is used. In this case, if the UID or 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: **Before you begin:** You need to know which RACF group profiles will be used as z/OS UNIX groups.

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

1. Log on to the 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: `ADDGROUP OMVSGRP SUPGROUP(SYS1) OWNER(SYS1) OMVS(GID(575))` **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:
	`ALTGROUP ENGNGP7 OMVS(GID(678))`
	
	Result: You have created a z/OS UNIX group.
	
	Tips:
	• Use AUTOGID to automatically assign an unused GID.
	Example:
	`ALTGROUP ENGNGP7 OMVS(AUTOGID)`
	• 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."
- Using the BPX.SUPERUSER resource in the FACILITY class. See "Using the BPX.SUPERUSER resource in the FACILITY class" on page 26.
- Assigning a UID of 0, which is the least desirable way. See "Assigning a UID of 0" on page 29.

  For specific installation requirements regarding superuser authority, see "Security requirements for ServerPac and CBPDO installation" on page 34.

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 1 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 1. 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 specific users or groups. **Note:** User/group must have the same access that would be required to SUPERUSER.FILESYS while accessing the file	See note.
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

Table 1. Resource names in the UNIXPRIV class for z/OS UNIX privileges (continued)

Resource name	z/OS UNIX privilege	Minimum access required
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
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

Table 1. Resource names in the UNIXPRIV class for z/OS UNIX privileges (continued)

Resource name	z/OS UNIX privilege	Minimum access required
Notes: 1. See “Steps for setting up the CHOWN.UNRESTRICTED profile” on page 25. 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 1 on page 22, except the privilege associated with the CHOWN.UNRESTRICTED resource (see “Steps for setting up the CHOWN.UNRESTRICTED profile” on page 25).

Steps for assigning selected users to transfer ownership of any file: **Before you begin:** You need to know which users will be assigned superuser authority.

Perform the following steps to 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.

   ```
   SETROPTS CLASSACT(UNIXPRIV)
   ```

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

---

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

   ```
   SETROPTS RACLIST(UNIXPRIV)
   ```

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

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

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

Allowing all 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.

Steps for setting up the CHOWN.UNRESTRICTED profile: **Before you begin:** You need to know that 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.

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.

   ```
   SETROPTS CLASSACT(UNIXPRIV)
   ```

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

Allowing selected z/OS UNIX users to transfer file ownership 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 24 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 the **su** command to obtain superuser authority. It is assumed that the default group for SYSPROG is set up with a GID.

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

When you are done, you have set up the BPX.SUPERUSER resource in the FACILITY class and permitted the users 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

Give each user a unique UID and have them use the **su** command to obtain the authority they need. You can give them the ability to use the **su** command by giving them READ authority to the BPX.SUPERUSER resource in the FACILITY class. For more information 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: **Before you begin:** You need to know which superusers you want to change from UID(0) to a unique nonzero UID.

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

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

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`

If you choose this method . . .	Then . . .
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 owning UID of the **/u/john** directory is changed to 7. The owning UID of all files and subdirectories of the **/u/john** directory is also changed.

   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 the section about customizing $HOME/.profile in the “Customizing the shells and utilities” chapter.

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.

Tip: 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 the SHARED.IDS profile is defined in the UNIXPRIV class, 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

You can control who can use certain UNIX functions when you define RACF profiles with UACC(NONE) to protect the appropriate BPX.* resources in the FACILITY class. Generally, authorized users need at least READ access to the BPX.* resources in order to use the UNIX function.

To activate RACF control of UNIX functions, use the RACF SETROPTS CLASSACT FACILITY command. Permit your authorized users to the appropriate resources before you activate the FACILITY class or else users will be not be able to use protected UNIX functions.

For security reasons, you may need to define these FACILITY class profiles:

- **BPX.CONSOLE**

  Allows a permitted user the ability to use the _console() or _console2() services.
- **BPX.DAEMON**

  BPX.DAEMON serves two functions in the z/OS UNIX environment:
  - Any superuser 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 FACILTY class profile is defined, then z/OS UNIX will verify that the address space has not loaded any executables that are uncontrolled before it allows any of the following services that are controlled by z/OS UNIX to succeed:

- seteuid
- setuid
- setreuid
- pthread_security_np()
- auth_check_resource_np()
- _login()
- _spawn() with user ID change
- _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 the section on establishing the correct level of security in the "Setting up for daemons" chapter.

- **BPX.DAEMON.HFSCTL**

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

- **BPX.DEBUG**

  Users with READ access to 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 a z/OS UNIX file. This authority allows the user to create a program that will run APF-authorized. This is similar to the authority of allowing a programmer to update SYS1.LINKLIB or SYS1.LPALIB.

- **BPX.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 Administrator's Guide* has more information about BPX.NEXT.USER.

- **BPX.FILEATTR.PROGCTL**

  Controls which users are allowed to set the program control attribute. Programs marked with this attribute can execute in server address spaces that run with a high level of authority. See the section on defining UNIX files to program control in the "Setting up for daemons" chapter 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 the section on defining UNIX files as shared programs in the "Setting up for daemons" chapter.

- **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 the section on fastpath support for system authorization facility (SAF) in the “Managing processing for z/OS UNIX” chapter.

- **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:
  - The “Preparing security for servers” chapter
  - The section on establishing the correct level of security for daemons in the “Setting up for daemons” chapter

- **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 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 20.

- **BPX.UNLIMITED.OUTPUT**

  Allows users to use the BPX_UNLIMITED_OUTPUT environment variable to override the default spooled output limits for processes.

- **BPX.WLMSERVER**

  Controls access to the WLM server functions _server_init() and _server_pwu(). It also controls access to these C language WLM interfaces:

- QuerySchEnv()
- CheckSchEnv()
- DisconnectServer()
- DeleteWorkUnit()
- JoinWorkUnit()
- LeaveWorkUnit()
- ConnectWorkMgr()
- CreateWorkUnit()
- ContinueWorkUnit()

A server application with read permission to this FACILITY class profile can use the server functions, as well as the WLM C language functions, to create and manage work requests.

Permissions for the FACILITY class profiles

Figure 1 on page 34 lists whether the caller is permitted to use the services with the indicated profile if that profile is defined and if the caller's user ID is permitted to the specified RACF FACILITY class profile.

- YES indicates that the caller is permitted to use the services associated with the profile.
- NO indicates that the caller is not permitted to use the services associated with the profile

For example, if BPX.DAEMON is not defined and the caller has a UID of 0, then that caller would be permitted to use setuid. However, if BPX.DAEMON is defined and the caller is permitted to it but has a nonzero UID, then that caller would not be permitted to use setuid.

	Profile is not defined		Profile is defined			
	(not applicable)		Not Permitted		Permitted	
	If UID(0)	If not UID(0)	If UID(0)	If not UID(0)	If UID(0)	If not UID(0)
BPX.CF	No	No	No	No	Yes	Yes
BPX.CONSOLE (1)	Yes	No	Yes	Yes	Yes	No
BPX.DAEMON	Yes	No	No	No	Yes	No
BPX.DAEMON.HFSCTL	No	No	No	No	Yes	Yes
BPX.DEBUG	No	No	No	No	Yes	Yes
BPX.FILEATTR.APF	No	No	No	No	Yes	Yes
BPX.FILEATTR.PROGCTL	No	No	No	No	Yes	Yes
BPX.FILEATTR.SHARELIB	No	No	No	No	Yes	Yes
BPX.JOBNAME	Yes	No	Yes	No	Yes	Yes
BPX.MAINCHECK	No	No	Yes	Yes	Yes	Yes
BPX.MAP	Yes	No	No	No	Yes	Yes
BPX.NEXT.USER (2)	- -	- -	- -	- -	- -	- -
BPX.UNLIMITED_OUTPUT	Yes	No	Yes	No	Yes	Yes
BPX.POE	Yes	No	No	No	Yes	Yes
BPX.SAFFASTPATH	No	No	No	No	Yes	Yes
BPX.SERVER	Yes	No	No	No	Yes	Yes
BPX.SHUTDOWN	Yes	No	No	No	Yes	Yes
BPX.SMF	No	No	No	No	Yes	Yes
BPX.SRV.userid (3)	No	No	No	No	Yes	Yes
BPX.STOR.SWAP	Yes	No	No	No	Yes	Yes
BPX.SUPERUSER	No	No	No	No	Yes	Yes
BPX.WLMSERVER	Yes	No	No	No	Yes	Yes

Note:

1. The BPX.CONSOLE profile is used to control access to authorized features of the _console() service and is not used to control which users can use the base _console() service.
2. The BPX.NETXT.USER profile is used by RACF to assign UIDs and GIDs when creating or altering a user ID's OMVS segment and is not processed directly by z/OS UNIX.
3. BPX.SRV.userid profiles are defined in the RACF SURROGAT class.

Figure 1. Permissions for defined and undefined FACILITY class profiles

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 resources in the 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.** Doing so 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 36
- "If you have problems with names such as UUCP, UUCPG, and TTY" on page 36

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 (*uupcg*) do not conflict with existing names, perform the steps for uppercase IDs in “If you use uppercase group and user IDs” on page 35.

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 the section about USERIDALIASTABLE in the “Customizing z/OS UNIX” chapter.

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 the section about USERIDALIASTABLE in the "Customizing z/OS UNIX" chapter.

   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
   XXTTY        TTY
   XXUUCPG  uucpg
   :userids
   XXUUCP     uucp
   ```

5. Activate the user ID alias table. If you are already using the user ID alias table, new database queries will yield the new alias if the userid performing the query has read/execute access to the userid/group name alias table. The table is checked every 15 minutes and refreshed if it has been changed. If a change needs to be activated sooner, you can use the SETOMVS or SET™ OMVS operator commands.

   If you are not using the user ID alias table, you can use the SET OMVS operator command to activate it now.

   Example:

   ```
   SET OMVS USERIDALIASTABLE=/etc/tablename
   ```

   where **/etc/tablename** is the name of your user ID alias table. You can also use the SETOMVS operator command.

6. Specify USERIDALIASTABLE in your BPXPRMxx parmlib member to make this change permanent for your next IPL.
7. Perform these tasks on all of your driving, test, and production system images.

For more information, see:

- "Defining z/OS UNIX users to RACF" on page 7
- "Creating z/OS UNIX groups" on page 19
- *z/OS MVS System Commands*
- *z/OS Security Server RACF Administrator's Guidez/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 9.

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 43 for more information about ACLs.

Setting classes for a user’s process

The access permission bits are set for three classes. When a user’s process accesses a file, the system determines the class of the process and then uses the permission bits for that class to determine if the process can access the file. For a file, a process can be in only one class. The class for a process can be different for each file or directory.

The class is one of the following:

- **Owner class:** Any process with an effective UID that matches the UID of the file.
- **Group class:** Any process with an effective GID or supplemental group GID that matches the GID of the file when the UIDs do not match.
- **Other class:** Any process that is not in the owner or group class, such as when the UIDs or GIDs do not match.

By default, the system sets the UID and GID of the file when the file is created:

- The UID is set to the effective UID of the creating process.
- The GID is set to the GID of the owning directory. You can define FILE.GROUPOWNER.SETGID to change this behavior; see "Steps for setting up the FILE.GROUPOWNER.SETGID profile."

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 21.

If you want to specify that, by default, the group owner of a new file is to come from the effective GID of the creating process, you need to set up a profile in the UNIXPRIV class called FILE.GROUPOWNER.SETGID. "Steps for setting up the FILE.GROUPOWNER.SETGID profile" describes the process.

Steps for setting up the FILE.GROUPOWNER.SETGID profile: Perform the following steps to set up the FILE.GROUPOWNER.SETGID profile.

1. Define the FILE.GROUPOWNER.SETGID profile.

   ```
   RDEFINE UNIXPRIV FILE.GROUPOWNER.SETGID
   ```

   ---

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

   ```
   SETROPTS CLASSACT(UNIXPRIV)
   ```

   ---

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

   ```
   SETROPTS RACLIST(UNIXPRIV)
   ```

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

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

When you are done, you have set up the FILE.GROUPOWNER.SETGID profile. The set-gid bit for a directory determines how the group owner is initialized for new objects created within the directory.

- If the set-gid bit is on, then the owning GID is set to that of the directory.
- If the set-gid bit is off, then the owning GID is set to the effective GID of the process.

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 2 on page 40 shows types of access and the permissions granted by the accesses.

Table 2. 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 path names of files the user should use. Read permission is required for some options of some commands.
- Write permission to all directories in which the user will be creating 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 44).
- 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 43.

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). SAF services handle the FSP (and ACL) maintenance. z/OS UNIX provides commands that invoke these SAF services.

For information about using RACF authorization to grant privileges for use of local files and directories, see Table 1 on page 22.

Obtaining security information for a file

Users with search access to the directories in the path name and, for some options, read access to the directories can check a file’s security information, including the access permissions. They do not need read access to the file being checked. Programs can also check security information for files.

To check the security information, do one of the following:

- Use the ISPF shell
- Enter the **ls -l** or **ls -E** shell command.
- Run a stat() or fstat() 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
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 S	Third (owner only) Third (owner only)	Execute permission for owner, set-user-ID set No execute permission for owner, set-user-ID set
s S	Third (group only) Third (group only)	Execute permission for group, set-group-ID set No execute permission for group, set-group-ID set
t T	Third (other only) Third (other only)	Execute permission for other, sticky bit set 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, zFS, and TFS file systems support ACLs. It is possible that other physical file systems will eventually support z/OS ACLs. Consult your file system documentation to see if ACLs are supported.

Before you can begin using ACLs, you must know what security product is being used. The ACLs are created and checked by RACF, not by the kernel or file system. If a different security product is being used, you must check their documentation to see if ACLs are supported and what rules are used when determining file access.

Notes®:

1. The phrases “default ACL” and “model ACL” are used interchangeably throughout z/OS UNIX documentation. Other systems that support ACL have default ACLs that are essentially the same as the directory default ACLs in z/OS UNIX.
2. According to the X/Open UNIX 95 specification, additional access control mechanisms 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 46 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 on page 44, 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. 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/joeadmn/Admins** contains a list of ACL entries for users and groups who need to support some administrative work. The file contains ACL entries, one per line, in the format that **setfacl** expects and which **getfacl** displays. These people must be granted access to all of the directories within the file system subtree starting and including **/admin/work**.

   ```
   setfacl -S /u/joeadmn/Admins $(find /admin/work -type d)
   ```

   This example uses shell command substitution to use the output of the **find** command as input to the **setfacl** command. The **/u/joeadmn/Admins** file 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.

Guideline: 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).

Guideline: Analyze your file system space utilization before implementing default ACLs in your file system. If you use both file and directory default ACLs in every directory in the file system, a separate physical ACL is created for every new file and directory. Using an access ACL for every directory will probably not cause concerns about space utilization. However, the same cannot be said of files, especially if the inherited ACLs are large.

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 3 summarizes the tasks that you might want to do and their associated commands.

Table 3. 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 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**

Table 3. ACL tasks and their associated commands (continued)

Task	Shell Command
Test files and directories for extended ACL information. Also test for directory ACLs and file default ACLs on directories.	**filetest**, **test**, [...] and [[...]] reserved-word command

How ACLs are used in file access checks: The algorithm for access checking is up to the security product that is being used. If the physical file system supports ACLs, then it uses the SAF ck_access (IRRSKA00) callable service when passing the ACL to the security product.

If the security product supports ACLs, it applies its own rules to the file access request. RACF uses the permission bits, access ACL, and various UNIXPRIV class profiles to determine whether the user is authorized to access the file with the requested access level. 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 a physical file system or zFS aggregate is created, the file system root will be assigned the security label that is specified in the RACF data set profile that covers the data set name. If a security label is not specified or if a data set profile does not exist, then a security label will not be assigned to the file system root.
- zFS file systems support the **chlabel** utility which allows the setting of an initial security label on a file or directory. 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, no label is assigned to the new object.
  - If the parent directory is assigned a security label other than SYSMULTI, the new file or directory is assigned the same security label as the parent directory.
- The rules for security label assignment are more extensive when running in a multilevel-secure environment. For more information, see *z/OS Planning for Multilevel Security and the Common Criteria*.

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 path name.

The character strings '$SYSSECA/' and '$SYSSECR/' have special meaning to z/OS UNIX when they appear as the first nine characters in a symbolic link. For more information on the use of these strings, see the section on assigning home directory and initial programs, depending on security labels, in *z/OS Planning for Multilevel Security and the Common Criteria*.

Using multilevel security

Multilevel security is a security policy that allows the classification of data and users on a system of hierarchical security levels combined with a system of non-hierarchical security categories.

In a multilevel-secure z/OS UNIX environment, security labels are used as described in "Using security labels" on page 48. To set the security label on z/OS UNIX files and directories, use the **chlabel** utility.

Restrictions:

- If you are using a shared file system, mixed sysplex support is not provided for multilevel security. Running a multilevel-secure environment in a mixed sysplex (with systems below V1R5) may cause 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 and the Common Criteria*.

Auditing access to files and directories

The security auditor uses reports formatted from RACF system management facilities (SMF) records to check successful and failing accesses to kernel resources. An SMF record can be written at each point where the system makes security decisions.

Six classes are used to control auditing of security events. These classes do not have any profiles. They do not have to be active to control auditing. Use the SETROPTS command to specify the auditing options for the classes. For a list of the classes used for auditing and an explanation of how to specify the audit options, see *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 setting audit controls, 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 path names and program names that are sanctioned by the installation for use by APF-authorized or program controlled calling programs. This file contains properly constructed path names and program names as defined in *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 dlload), exec, spawn, or attach_exec from an authorized program.

:programcontrol_path
This keyword is the start of a list of directories that is only used in the execution of an hfsload (or C dlload), 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

Restriction: You cannot use symbolic links (for example, $SYSNAME) in sanction lists. They will not work.

You have to follow certain formatting rules when creating sanction lists.

- Only use absolute path names.
- Path names cannot start with **/***.
- Each list element must be on a line by itself, with no comments. Lines are terminated with the newline character, as is consistent with the stepliblist and useridaliastable files. Leading blanks may be on the list element line and are ignored. Use the newline character to delimit a path name. Trailing blanks are ignored. Other white space is considered part of the path name.
- Follow standard z/OS UNIX path naming conventions.
- You must follow standard MVS program naming conventions.
- Encode the sanction list file in the IBM-1047 code page.
- You can include comment lines in the list. Each comment line must start with **/*** and end with ***/**. They cannot be on the same line with any other type of line.
- Do not enclose the path names or program names in quotation marks.

The tags **:authprogram_path**, **:programcontrol_path**, and **:apfprogram_name** must be used to delineate between the different types of sanction lists.

- If there are no tags in the file, then all data in the file is ignored and you will get a parsing error. If a tag is missing, then the subsequent processing of hfsload/dlload, exec or spawn will not change, based on the tag that was missing. The effect of different sanction lists is not cumulative. Once a sanction list is parsed and accepted, the contents provide the only active lists of path names and program names for hfsloads, execs, and spawns.
- List elements (path names or program names) before a tag are ignored.
- Lines after the last valid entry line (such as a path name or a program name) are ignored.
- If an **:authprogram_path** tag is present, then all lines following it and up to the next tag are considered to be approved path names from which authorized programs can be invoked.
- If a **:programcontrol_path 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 path names 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 path names and program names as you go along. A partially complete file can be activated and when addtional entries are known, this file can be updated. A background task will automatically check this file every 15 minutes for updates and then incorporate them.

You also need to be aware that only one sanction list check is done for each program invocation. Although links in directories are supported, sanction list processing only performs one check. This check uses the path name or program name that was specified by the user.

Tip: The installation can construct the sanction lists with link path names or actual path names, or both. The decision depends on how the site would like the users to invoke the programs. For example, if the actual directory is in the sanction list instead of the directory that contains the link, and the associated program is invoked via the link, the program would not be executed. The program is only executed if the directory where the link was defined or resides is specified in the sanction list and the associated program is invoked via the link. Alternatively, both the actual directory and directory where the link resides could be placed in the sanction list. This 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 path names and not links.

Perform the following steps to create a sanction list.

1. Create a sanction list, following the rules listed in “Formatting rules for sanction lists” on page 51. You can cut and paste the following sample.

```
 /*****************************************************************/
/*                                                               */
/*   Name: Sample authorized program list                        */
/*                                                               */
/*   Description:  Contains lists of approved directories and    */
/*                 program names from which privileged programs  */
/*                 may be invoked                                */
/*                                                               */
/*****************************************************************/
/*****************************************************************/
/* Authorized program directories                                */
/*****************************************************************/
:authprogram_path
/bin/test
/bin/test/beta

/*****************************************************************/
/* Program control directories                                   */
/*****************************************************************/
:programcontrol_path
/in/test/specials
```

```
/*****************************************************************/
/* APF authorized programs                                       */
/*****************************************************************/
:apfprogram_name
PAYOUT
```

2. Give the sanction list a name.

   Guideline: The path name of the sanction file should be **/etc/authfile**, in keeping with IBM's strategy to place all customized data in the **/etc** directory.

When you are done, you have created a sanction list and named it. To activate it, see "Steps for activating the sanction list."

Rule: Only users with superuser authority should be given update access to sanction lists.

Steps for activating the sanction list

Before you begin: You must know what the file name is for your sanction list. This file may or may not exist, or it may not be complete, or both. If this file exists, it must be properly constructed as described in "Formatting rules for sanction lists" on page 51 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 AUTHPGMLIST statement in BPXPRMxx. The sanction list may or may not have already been set up.	Customize the BPXPRMxx parameter. **Example:** `AUTHPGMLIST('/etc/authfile')`

Use SETOMVS.
 	
Guideline: You should already have set up the sanction list. Otherwise, you will get an error message warning you that the file does not exist. The path name, however, will be set. If you issue the same command with the same file name, you will not get an error message. The DISPLAY OMVS command will show the AUTHPGMLIST parameter being set. This file name is used by the background task to check whether a sanction list has been created or updated.
	Issue the SETOMVS command.
	
	Example:
	`SETOMVS AUTHPGMLIST='/etc/authfile'`
	
	Tip: To turn off sanction list checking, issue:
	`SETOMVS AUTHPGMLIST=NONE`

If you choose this method . . .	Then . . .
A nonexistent sanction list. **Guideline:** Use this feature only if the sanction list must not exist before it is activated. It is possible to set the sanction list value and forget that the sanction list has not been completely set up. The system 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 occur for spawns, execs, and so on. This sanction list file must be set up for sanctioning to occur. The background task will not warn that the sanction list does not exist.	Use either method described in this table (customize the BPXPRMxx parmlib member or use SETOMVS).

2. If the sanction list has not already been created (see “Steps for creating a sanction list” on page 52), 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 AUTHGPGMLIST 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 7.
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. Two levels of privileges are available for servers and daemons: UNIX level and z/OS UNIX level. In providing security, you need to understand the differences between the two levels. For discussions of the levels, see:

- The section on establishing the correct level of security for daemons in the *Setting up for daemons* chapter.
- The section on establishing the correct level of security for servers in the *Preparing security for servers* chapter.

Tuning performance

Overview of tuning performance

You need to take some tuning steps because you are combining MVS and UNIX. Two tuning situations exist, depending on how your system is being used: as a production system or a porting system. For both, you can take important steps to improve performance and control resource consumption.

Guideline: If your system is running in a virtual server or as a VM guest, the storage size should be at least 64M.

To learn how to improve performance on a porting system, read Chapter 8 of *Porting Applications to the z/OS UNIX Platform*.

In this chapter

This chapter covers the following subtasks.

Subtasks	Associated procedure (see . . .)
Caching UID and GID information in VLF	"Steps for caching UID and GID information in VLF" on page 58
Moving an executable in the file system into the LPA	"Steps for moving an executable in the file system into the LPA" on page 59
Setting process limits	"Steps for setting process limits in z/OS UNIX" on page 77
Changing process limits	"Steps for changing the process limits for an active process" on page 81

Using DASD cache

Place on cached DASD:

- Volumes that contain user file systems. In most cases, writes to the HFS or zFS file system will not cause delays waiting for disk I/O. However, it is still wise to place file system data sets on fast, cached DASD with DASD FAST WRITE turned on.

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 “Organizing file systems to improve performance” on page 71 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 z/OS UNIX.

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 modified 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.

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 in STEPLIBs

If you decide not to put the run-time library in the link list, 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 z/OS XL C/C++ compiler and Program Management Binder in the LPA.

Putting compiler load modules into dynamic LPA

On systems where application development is the primary activity, performance may benefit if you put the compiler data set CBC.SCCNCMP or load modules into the dynamic LPA by making these changes to the PROGxx member of SYS1.PARMLIB:

```
SYS1.PARMLIB(PROGxx)
 LPA ADD MASK(*),DSNAME(CBC.SCCNCMP)
```

or

```
LPA ADD MODNAME(CCNDRVR,CCNECICS,CCNEDFLT,CCNEDSCT,
    CCNEDWRT,CCNED170,CCNEFILT,CCNEHIFC,CCNEIPAO,
    CCNEIPA1,CCNEIPA2,CCNEMDEP,CCNEOPTP,CCNEP,CCNEPP,
    CCNER,CCNETBY,CCNMSGE,CCNMSGK,CCNMSGT,CXXFILT,
    ICCNMSGT),DSNAME(CBC.SCCNCMP)
```

All compiler modules run above the line and they consume just over 142 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 EDCRNLST)	Above the line

Caching RACF user and group information in VLF

Caching UIDs and GIDs improves performance for commands such as **ls -l**, which must convert UID numbers to user IDs and GID numbers to RACF group names. RACF allows you to cache UID and GID information in Virtual Lookaside Facility (VLF).

Steps for caching UID and GID information in VLF

Before you begin: You need to have access to the COFVLxx member of SYS1.PARMLIB.

Perform the following steps to cache UID and GID information in VLF.

1. Add these VLF options to the COFVLFxx member of SYS1.PARMLIB.

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

2. Start VLF, specifying the updated member.

   Example: In this example, the updated member is COFVLF33.

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

When you are done, you have cached GID and UID information in VLF.

Because VLF is started after RACF and OMVS, you 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 the section on using the COFLFxx parmlib member to activate RACF classes in the *Customizing z/OS UNIX* chapter.

Moving executables into the link pack area

Some executables 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().

Guideline: One thing to consider when you analyze which executables belong in LPA is that modules with the sticky bit on are not eligible for local spawn(). If your executable is normally invoked by spawn(), either by the shell or by another application, turning on the sticky bit forces spawn() processing to execute the program in a spawned child address space. In cases where local spawn() would be used if the sticky bit were not on, this reduces the benefit of loading the executable from the LPA.

Steps for moving an executable in the file system into the LPA

Before you begin: You need to know how long the executable or DLL name is.

Perform the following steps to move an executable in the file system into the LPA.

1. Select one of the following actions, depending on how long the executable or DLL name is.

If . . .	Then . . .
The executable or DLL name is no more than 8 characters excluding the extension (such as longname.dll) and contains no special characters that are not valid for TSO PDSE member names.	1. Bind the executable or DLL into a PDSE member (for example, LONGNAME) 2. For the executable or DLL in the file system, turn on the sticky bit. **Example:** `chmod +t longname.dll` 3. Verify that the executable is marked reentrant. **Tip:** You can check that the executable is marked reentrant by checking TSO browse on the PDSE, locating the member in the member list, pressing PF11 and then looking for the RN attribute. 4. Put the executable into dynamic LPA by modifying the PROGxx parmlib member or by issuing the SETPROG console command.

The executable or DLL name is more than 8 characters long, excluding the extension (for example, reallylonglongname.dll), or if the name contains special characters.	1. Bind the executable or DLL into a PDSE member with a valid member name (for example, REALLY)
	2. Rename the original executable or dll to save it.
	Example:
	`mv reallylonglongname.dll reallylonglongname.dll.save`
	3. Create an external link for the name.
	Example:
	`ln -e REALLY reallylonglongname.dll`
	4. Verify that the executable is marked reentrant.
	Tip: You can check that the executable is marked reentrant by checking TSO browse on the PDSE, locating the member in the member list, pressing PF11 and then looking for the RN attribute.
	5. Put the executable into dynamic LPA by by modifying the PROGxx parmlib member or by issuing the SETPROG console command.

When you are done, you have moved an executable in the file system into the LPA.

Binding the executable or DLL into a PDSE

To bind the executable or DLL into a PDSE you can use sample JCL in Figure 2 on page 61. While this JCL will work for most simple executables, the binder options (specified as PARM= below) will not be appropriate for all executables or DLLs. If you have a Makefile for the executable or DLL, this will tell you what binder options should be used.

Because most executables in the file system today are program objects (new load module format), they must be bound into PDSE libraries. So, SYSLMOD DD should point to a PDSE (Data Set Name Type = Library).

```
//PUTINLPA  JOB  MSGLEVEL=(1,1)
//*                                                              *
//* INLMOD DD STATEMENT SPECIFIES THE DIRECTORY THAT CONTAINS  *
//* THE PROGRAM.                                                 *
//*                                                              *
//* THE INCLUDE STATEMENT SPECIFIES THE NAME OF THE FILE TO    *
//* RUN FROM THE LPA.                                            *
//*                                                              *
//* THE NAME STATEMENT SPECIFIES THE FILE NAME BUT IN          *
//* UPPERCASE. THIS MUST BE SAME AS THE FILE NAME.               *
//*                                                              *
//LINK     EXEC PGM=IEWL,REGION=100M,
//  PARM='LIST,XREF,LET,RENT,REUS,AMODE=31,RMODE=ANY,CASE=MIXED'
//SYSUT1   DD UNIT=SYSDA,SPACE=(CYL,(10,10))
//SYSPRINT DD SYSOUT=*
//INLMOD   DD PATH='/bin/'
//SYSLMOD  DD DSN=OECMD.LPALIB,DISP=SHR
//SYSLIN   DD *
         INCLUDE INLMOD(myprog)
         ENTRY   CEESTART
         NAME    MYPROG(R)
/*
```

Figure 2. Job for placing a program in the LPA

Use an SMP/E USERMOD to link any IBM-supplied programs from a UNIX file system into another library, such as when loading it into LPA. Doing so automatically keeps the two copies of the module at the same level when service is installed. It also provides a record of modifications to your systems. See *z/OS 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 about PROGxx, 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. The loadhfs section in *z/OS UNIX System Services Programming: Assembler Callable Services Reference* discusses both shared object library programs and the ST_SHARELIB extended attribute.

Executables that have the ST_SHARELIB extended attribute turned on are called *system shared library programs*. They are an optimal way of sharing large executables across many address spaces in the system. These executables are shared on a megabyte boundary to allow for the sharing of a single-page table (similar to LPA). The storage used in the user address space to establish the mapping to the shared library region is from the high end of private storage; in most cases, it does not interfere with the virtual storage used by the application program.

Guideline: The amount of storage that is carved out of the high end of private storage of each address space that loads a system shared library object is based on the value of the SHRLIBRGNSIZE parameter in the BPXPRMxx parmlib member. If this value is set too high, the storage set aside for the mapping of the shared library region may interfere with the private storage requirements of each of these address spaces. For this reason, the value specified for SHRLIBRGNSIZE should be the minimum size that is required to contain all of the shared library

programs that are to be used on the system. Note that z/OS UNIX attempts to map the entire SHRLIBRGNSIZE into the private region, not just the portion that contains programs. If the private region is too small to map the entire shrlibrgnsize, then this shared library regionwill not be used, but a message will not be issued to indicate that this has happened.

Executables that have the .so suffix in their file names are called *user shared library programs*. They are optimal for sharing 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. In most cases, it comes out of the same user region storage that is used for private area loaded modules. Once the ST_SHARELIB bit has been set for a module, any program, whether or not it has read access to the BPX.FILEATTR.SHARELIB FACILITY, will be able to load modules into the system shared library and use those that are already loaded.

See the section on defining UNIX files as shared library programs for information about setting the ST_SHARELIB extended attribute.

Tuning limits in BPXPRMxx

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 suggestions 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.

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)
```

Parameter	Initial setting	Note
MAXUIDS	20	If you allow 20 current TSO/E users to access the z/OS UNIX system, each of them could consume twice the resource they normally used for TSO/E. This would require all your remaining system resources.

Parameter	Initial setting	Note
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.

See the BPXPRMxx section in *z/OS MVS Initialization and Tuning Reference* for a complete description of each BPXPRMxx statement. For more details about ESQA and other storage requirements for MVS, see the section on evaluating virtual storage needs in the*Customizing z/OS UNIX* chapter.

Tip: Consider adjusting MAXSHAREPAGES on an active system. Dynamically decreasing the number of pages available to ESQA may cause errors because, for those jobs, the ESQA limit may be reached or exceeded. It is possible that shared programs will not be able to be loaded and fork() may not succeed. This situation will exist until the workload adjusts to the new lower limit.

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

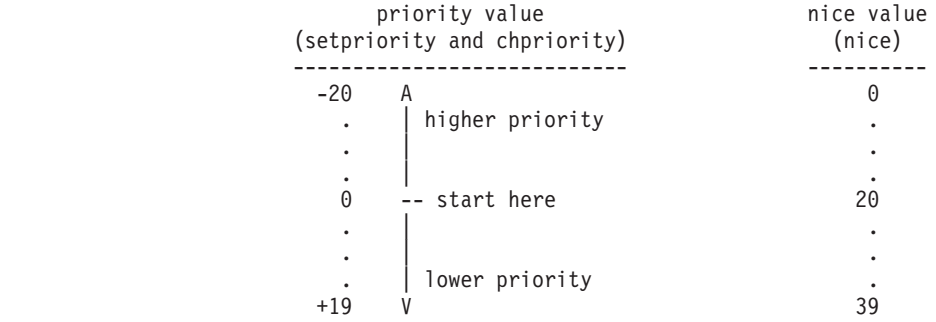
controls. However, nice(), setpriority(), and chpriority() support is provided. This support interfaces with SRM and workload manager to provide system control and monitoring support.

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.



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.

Guideline: Do not specify PRIORITYPG and PRIORITYGOAL in BPXPRMxx unless you need nice() and setpriority() support. It is simplest and best to give MVS full control over priorities of work.

Tuning tips for SYS1.PARMLIB

This section contains tuning tips that involve updating SYS1.PARMLIB.

Putting the SCEELPA data set in the LPA list

Because the SCEERUN data set has many modules that are not reentrant, you cannot place the entire data set in the link pack area using the LPALIST member of SYS1.PARMLIB. However, you can take advantage of a SCEELPA data set that contains a subset of the SCEERUN modules – those that are reentrant, reside above the line, and are heavily used by z/OS UNIX.

To improve performance, put the SCEERUN data set in the link list (LNKLSTxx member). Then place the new SCEELPA data set in LPA list, using the LNKLSxx member of SYS1.PARMLIB.

Tip: You can also add additional modules to the LPA, using the dynamic LPA capability (SET PROG=). This method is preferable to adding modules to the LPA by using the Modify Link Pack Area (MLPA=) option at IPL, because it avoids the performance degradation that occurs with the use of MLPA.

For more information about LPALSTxx and LNKLSTxx, see *z/OS MVS Initialization and Tuning Reference*.

Putting the linkage editor into dynamic LPA

Rule: Your primary Language Environment level must be the default level for the release, and you must be using the default compiler. To verify your Language Environment primary level, check that the library name (for example, CEE.SCEERUN) appears first in the linklist concatenation in the LNKLSTxx member of SYS1.PARMLIB.

On systems where application development is the primary activity, performance may benefit if you put the linkage editor into the dynamic LPA by making these changes to the PROGxx member of SYS1.PARMLIB:

```
SYS1.PARMLIB(PROGxx)
 LPA,ADD,MODNAME(CEEBINIT,CEEBLIBM,CEEEV003,EDCZV),DSNAME(CEE.SCEERUN)
 LPA,ADD,MODNAME(IEFIB600,IEFXB603),DSNAME(SYS1.LINKLIB)
```

Putting CBC.SCCNCMP in the LPALIST concatenation

Rule: Your primary Language Environment level must be the default level for the release, and you must be using the default compiler. To verify your Language Environment primary level, check that the library name (for example, CEE.SCEERUN) appears first in the linklist concatenation in the LNKLSTxx member of SYS1.PARMLIB.

On systems where application development is the primary activity, performance may benefit if you put CBC.SCCNCMP in the LPALST concatenation. All modules run above the line, and they consume just over 42 MB.

Ensuring VLF caching of UIDs and GIDs

Rule: Your primary Language Environment level must be the default level for the release, and you must be using the default compiler. To verify your Language

Environment primary level, check that the library name (for example, CEE.SCEERUN) appears first in the linklist concatenation in the LNKLSTxx member of SYS1.PARMLIB.

To ensure VLF caching of UIDs and GIDs, add the following statements to your COFVLFxx parmlib member if they are not already included:

```
...
 ...
 CLASS NAME (IRRGMAP)
       EMAJ (UMAP)
 CLASS NAME (IRRUMAP)
       EMAJ (UMAP)
 CLASS NAME (IRRGTS)
       EMAJ (GTS)
 CLASS NAME (IRRACEE)
       EMAJ (ACEE)
```

After you are done, make sure that you have VLF started and are using the COFVLFxx parmlib member.

Tuning tips for the file system

You can use the following tips to fine-tune your file system:

- Make sure that all files in your file system have valid owning UIDs and GIDs. If you restore files from an archive and accidentally keep a UID and GID that were valid on another system, it can create problems that affect response time. For example, say that there is an invalid UID associated with a file. When you use a utility that checks the UID (such as **ls -l**), RACF searches the entire database for the UID.
- Place HFS data sets on volumes that are cached with DASD Fast Write.
- Give each user a separate mountable file system. Doing so enables you to spread user file systems across multiple DASD devices, to avoid I/O contention.
- Use control unit caching with DASD Fast Write. Doing so can improve **make** processing.
- Use the temporary file system (TFS) for **/tmp**.

Tuning tips for the compiler utilities

This section contains tuning tips for the compiler utilities (**c89**/**cc**/**cxx**/**c++**).

Restriction: The main compiler executable (**c89**/**cc**/**cxx**/**c++**) cannot be made a shared library object because it is not a DLL that is loaded. Instead, it is executed via spawn or exec.

If you are using the compiler utilities, you have two tuning options:

- Turn off the **s** extended attribute as described in “Turning off the s extended attribute for the compiler utilities.” The extended attribute is turned off because the computer utilities should not be locally spawned.
- Put the **c89**, **cc**, **cxx** and **c++** commands into the LPA. “Putting the compiler utilities into the LPA” on page 68 describes the procedure.

Turning off the s extended attribute for the compiler utilities

Because the compiler utilities should not be locally spawned, use the **extattr** command with the **-s** option to turn off the extended attribute for these utilities. The _BPX_SHAREAS and _BPX_SHAREAS_REUSE environment variables will then be ignored during spawn. You will also gain most of the performance benefit of putting the module in LPA with less effort.

Rule: To specify the **–s** extended attribute, you must be the owner of the file or have superuser authority.

Example:

```
extattr -s /bin/c89 /bin/cc /bin/cxx /bin/c++
```

However, if you want the best compile performance, put the compiler utilities in LPA as described in "Putting the compiler utilities into the LPA." Like using the **–s** extended attribute, putting them in LPA prevents local spawn and also gives you the other benefits of putting modules in LPA, such as no load from disk and one copy of the module in virtual storage.

Putting the compiler utilities into the LPA

In general, follow these steps when moving a module into the LPA, after making sure that the s extended attribute has not been set.

1. Preallocate a PDSE library using ISPF option 3.2. For a PDSE, you must specify DSNTYPE=LIBRARY.
2. Bind the executable into the PDSE by issuing the **cp** command in the z/OS UNIX environment.

   Example:

   ```
   cp -X myprog "//'OECMD.LPALIB(MYPROG)'"
   cp -X mylongprogram "//'OECMD.LPALIB(MYLONGPG)'"
   ```
3. Add the PDSE library to the dynamic LPA. You have two choices:
   a. Use the SETPROG console command to add the library to the dynamic LPA without re-IPing.
   b. Update the PROGxx parmlib member to ensure that the library is in the dynamic LPA after the next IPL.
4. Point to the new program by using an external link from the z/OS UNIX environment.

   Example:

   ```
   mv myprog myprog.backup
   mv mylongprogram mylongprogram.backup
   ln -e MYPROG myprog
   ln -e MYLONGPG mylongprogram
   ```

Steps for putting the compiler utilities into the LPA: **Before you begin:** You need to make sure that the s extended attribute has not been set. For more information, see "Turning off the s extended attribute for the compiler utilities" on page 67.

Perform the following steps to put the compiler utilities into the LPA.

1. Preallocate a PDSE library using ISPF option 3.2. For a PDSE, you must specify DSNTYPE=LIBRARY.

   _____

2. Bind **c89** into the PDSE. From the z/OS UNIX environment, issue the following command:

   ```
   cp -X /bin/c89 "//'OECMD.LPALIB(C89)'"
   ```

   _____

3. Add the PDSE library to the dynamic LPA. You have two choices:
   a. Use the SETPROG console command to add the library to the dynamic LPA without re-IPing.

b. Update the PROGxx parmlib member to ensure that the library is in the dynamic LPA after the next IPL.

---

4. Point to the compiler utilities, using external links. From the z/OS UNIX environment, issue the following commands:

```
cd /bin
mv c89 c89.backup
mv cc cc.backup
mv cxx cxx.backup
mv c++ c++.backup
ln -e c89 c89
ln -e c89 cc
ln -e c89 cxx
ln -e c89 c++
```

When you are done, you have put the compiler utilities into the LPA

Improving the z/OS shell performance

Rule: Do not specify PRIORITYPG and PRIORITYGOAL in BPXPRMxx unless you need nice() and setpriority() support. It is simplest and best to give MVS full control over priorities of work.

Tip: 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

You can use two environment variables to improve performance for the z/OS shell utilities: _BPX_SHAREAS and _BPX_SPAWN_SCRIPT. However, you cannot use them for the tcsh shell.

_BPX_SHAREAS: To improve performance in the z/OS shell, set _BPX_SHAREAS to YES. (REUSE is the same as 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().

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 ends, the child will end because it is a subtask.

If the extended attribute for the shared address space is not set, the program will not run in a shared address space, regardless of the setting of _BPX_SHAREAS. The attribute is set by **extattr +s** and reset by **extattr -s**. If the attribute is set, _BPX_SHAREAS has precedence.

_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 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, **/etc/profile** or **$HOME/.profile** should set the environment variable:

```
export _BPX_SPAWN_SCRIPT=YES
```

However, there may be exceptions, 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 3 and Figure 4 on page 71.

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 3, 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 3 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 3. 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, **$HOME/.profile** is different. Figure 4 on page 71 is an example of **$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 4. 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.

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.

Organizing file systems to improve performance

How well the file system performs depends on how it is organized. Because a mountable file system must reside on a single DASD volume, several file systems on a volume or too much activity in a single file system can cause DASD I/O response time to be a bottleneck. In addition, some file system locking is done on a mountable file system basis. For these reasons, each user should normally have a unique mountable file system.

Another consideration is the placement of files in the file system hierarchy. Files deep in the hierarchy require several lookups each time they are opened.

The /tmp directory

When you have a large number of interactive users, the **/tmp** directory can sustain large amounts of I/O activity. You can take any of these approaches:

- Mount a temporary file system (TFS) over **/tmp**, 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 the *Managing a temporary file system (TFS)* chapter.
- 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.

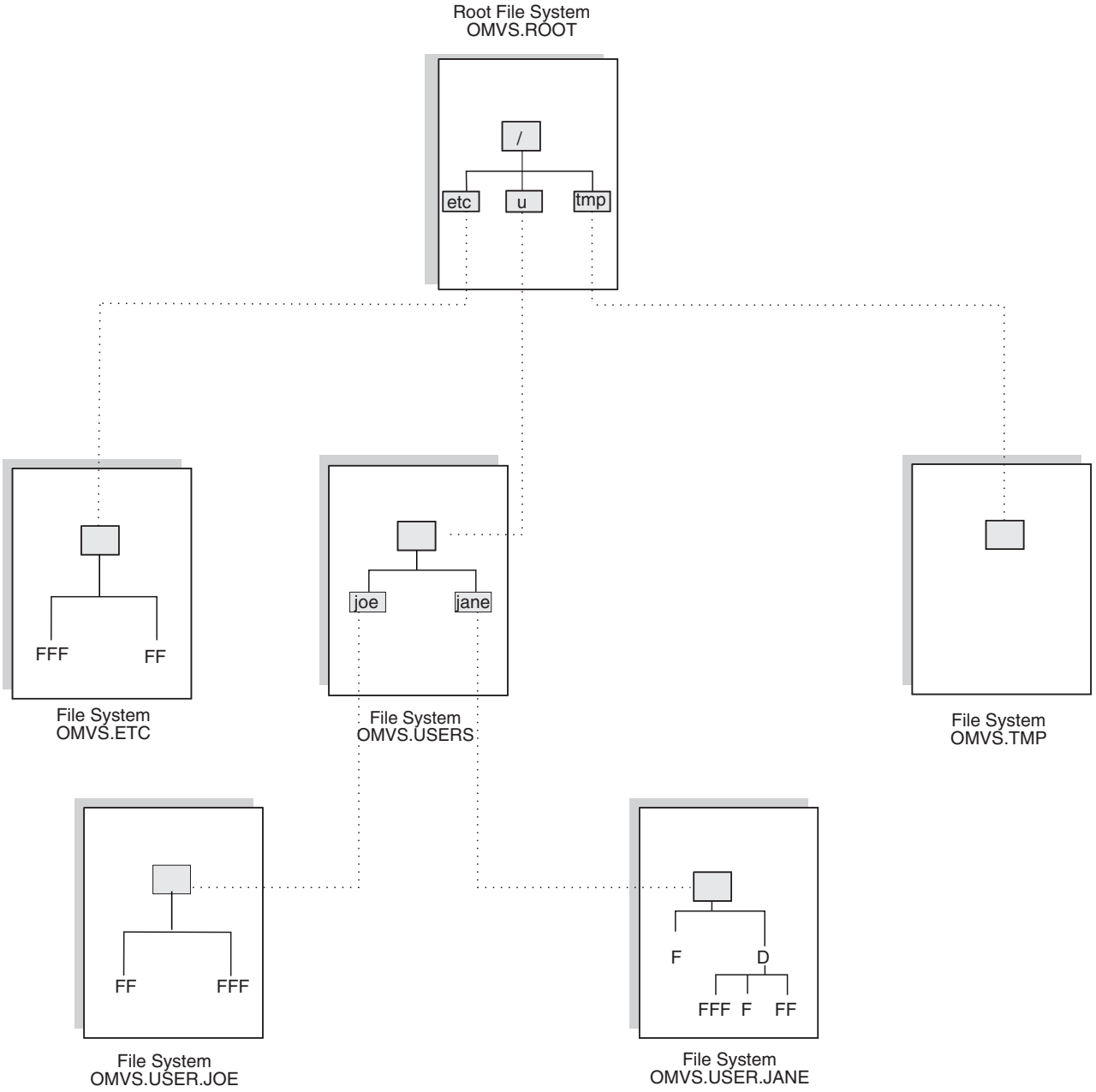


Figure 5. The /tmp directory on a mountable file system

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 the section about fastpath support for the system authorization facility (SAF) in the*Managing processing for z/OS UNIX* chapter.

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.

Handling process limits in z/OS UNIX

This section explains what system limits and process limits are and how to set process limits.

Understanding limits

Limits can be set for the system (system-wide limits) or for individual processes (process limits).

System-wide limits, which are limits that apply to every process, are set in the BPXPRMxx parmlib member. You can display limits defined in BPXPRMxx by using the operator commands D OMVS,OPTIONS and D OMVS,LIMITS.

- Limits associated with an MVS application derive their value from MVS. The soft and hard limit may be different when an MVS unit of work is dubbed.
- Limits for processes initiated by z/OS UNIX that cause an identity change, such as telnet, rlogin or a daemon process using setuid or exec, are created with the same hard and soft limits.

Process limits are limits that apply to individual processes. You can set some process limits by using the RACF user profile segment. Some process limits, such as the disk space, allow for a file or the size of a dump are set in BPXPRMxx.

- You can control the amount of resources used by specified z/OS UNIX users by setting individual limits for these users, as described in the section about setting limits for users in the *Establishing UNIX security* chapter. These limits apply to all users except for those with an UID of 0. Normally, when a process is initially dubbed, the soft limit is inherited from MVS and the hard limit is set in the BPXPRMxx parmlib member.

  For more information about UID(0) superuser authority, see the section on obtaining security information about users in the *Establishing UNIX security* chapter. Users with UID(0) will still have a limit but they can change it while other users can only change their soft limits.

- You can set limits on a process for resources such virtual memory space after you know which resources the application will need, and how the operating system affects application limits. For information about the types of processes and how they are created, see the section about types of processes in the *Introduction to z/OS UNIX* chapter.
- You can use the RACF ADDUSER and ALTUSER commands to specify the ASSIZEMAX limit on a per-user basis. See the section on limiting the use of memory objects in *z/OS MVS Programming: Extended Addressability Guide*, which discusses the use of MEMLIMIT. Table 6 on page 77 lists the process limits that you can set.

Before setting process limits as described in "Setting process limits in z/OS UNIX" on page 77, you need to understand how z/OS UNIX applications interact with the operating system and take into consideration all services that affect that resource. Then you can decide what soft and hard limits the application needs. For an explanation soft and hard limits, see "What are hard limits?" and "What are soft limits?."

What are hard limits?: The hard limit is the maximum value that a process's application can raise a soft limit to. The hard limit is derived from z/OS UNIX or RACF.

- Use BPXPRMxx statements to define z/OS UNIX defaults. Defaults exist for z/OS UNIX processes even when none are defined in BPXPRMxx. To find out what the defaults are, see the BPXPRMxx section in *z/OS MVS Initialization and Tuning Reference*.
- Limits defined in the RACF user profile. See Table 6 on page 77 for a list of hard limits that are defined in the RACF user profile.

z/OS UNIX mechanisms that affect limits are setrlimit(), inheritance from the parent and spawn inheritance structure (BPXYINHE), identity change, and dubbing. Any process can raise the soft limits to the hard limits. A superuser can raise both the hard and soft limits. A fork/spawn child inherits the same limits unless the parent changed the limits in the spawn inheritance structure or there is an identity change. The SETOMVS operator command can also affect these settings.

What are soft limits?: The soft limit is the value of the current process limit that is enforced by the operating system. If a failure such as an abend occurs, the application may want to temporarily change the soft limit for a specific work item, or change the limits of child processes that it creates. For example, a larger server daemon might reduce the amount of virtual memory available to a spawned child. New processes receive the same limits as the parent process as long as the installation or the application do not alter those values and an identity change does not occur. The soft limits for CPUTIME, ASSIZE and MEMLIMIT can be affected by several MVS limits mechanisms.

MVS limits are the soft limits provided to z/OS UNIX processes when z/OS UNIX services are invoked using TSO login, STC, or JCL. At the first request for a kernel service, the system dubs the program as a z/OS UNIX process. (For more information about z/OS UNIX applications, see the section on types of applications in the *Introduction to z/OS UNIX* chapter.) When a traditional MVS unit of work is first dubbed, the soft limits are normally obtained from MVS. The hard limit is normally obtained from BPXPRMxx if it is higher than the soft limit.

New processes that are created by a dubbed user receive the same soft and hard limits as the parent process if the installation has not changed the process limits and an identity change has not occurred.

How are limits handled after an identity change?: When the installation does not use any other method to define a limit, then exec and spawn handle limits after an identity change. After an identity change, an exec sets hard and soft limits to z/OS UNIX values. The change to z/OS UNIX values can also be accomplished by using a setuid() followed by an exec().

Spawn() can also cause an identity change. A spawned child that was created with a different identity than the parent using InheUserid or _BPX_USERID sets the hard and soft limits to the z/OS UNIX values for the new identity. A forked child with no identity change inherits the settings of the parent.

Task-level security (pthread_security or _login) causes spawn to use the limits of the new identity.

Resources values that processes receive when they are dubbed a process use the RACF profile to determine the hard limit if it is higher than the soft (current) limit. It is also used when processes are initiated by a daemon process using an exec after setuid(). In this case, both the RLIMIT_AS hard and soft limits are set to the address-space-size value.

Inheriting soft limits

Normally, soft and hard limits are not changed when a new process is created unless the creating process requests a change. Both the soft limits and the hard limits are set to the value that was specified the parent's inheritance structure, if it is requested. The values of both limits are raised to the hard limit if a child was spawned with a new identity, or if an exec occurred after the identity was changed.

Setting a global value in BPXPRMxx (using the MAXASSIZE parameter) and SMFPRMxx (using the MEMLIMIT parameter) provides the values the system will use to assign to all processes. The RACF security administrator may override these values by defining the MEMLIMIT value for individual identities in the associated RACF OMVS segment.

The only time MEMLIMIT or ASSIZE from the RACF OMVS segment is used to define these limits for inheritance is with a spawn with identity change or an exec after setuid. For more details, see "What happens when an identity change occurs?" below.

What happens when an identity change occurs?: When an identity change occurs, the new identity may or may not have an OMVS segment associated with it. If an identity change takes place, the following happens:

1. If the new identity has the limit defined by RACF, that limit overrides any other system control including the IEFUSI installation exit and the parent's inheritance structure.
2. If there is no RACF segment for the new identity, the IEFUSI installation exit may override the inheritance structure and set the limit.
3. If a RACF limit was not set for the new identity and if IEFUSI did not set the limit, the Inhe inheritance overrides the current values of the parent's inheritance structure.

4. The setting of the MAXASSIZE parameter in BPXPRMxx is the default for a new z/OS UNIX identity if none of the other values applied (except for MEMLIMIT, which is set in SMFPRMxx).

Setuid() alone or children created by either fork or spawn after a setuid retains the limits behavior of the previous identity.

What happens if an identity change does not take place when a child is created?: Normally, the child inherits the parent's current hard and soft limits. However, a change to the child's limit causes the child's hard and soft limit to be set to the hard limit. The child's limit can be changed by the parent in the inheritance structure on spawn() or by IEFUSI. How the system honors changes made in the inheritance structure or IEFUSI will vary depending on the parent's security setting.

When the same limit is not defined in the OMVS segment, the system will honor IEFUSI even if the parent requests a change using the inheritance structure (Inhe).

After a limit has been defined for a parent identity by the security administrator, that limit is trusted before other system components only if it is higher than the MVS limits. When a limit is defined in the OMVS segment:

- If the limit in the RACF OMVS segment is higher than the limit set in IEFUSI, the system ignores requests by IEFUSI to change the limit in the OMVS segment. The RACF OMVS segment is in control of ASSIZE and MEMLIMIT. If the RACF OMVS segment value is lower than IEFUSI, then IEFUSI sets hard and soft limits for both MEMLIMIT and ASSIZE, and IEFUSI is in control of ASSIZE and MEMLIMIT.
- If the RACF OMVS segment is in control of ASSIZE and MEMLIMIT for the parent, the system also ignores requests by IEFUSI to change limits for child processes that were created by that identity. In this case, the child is created with the parent's current limits. If IEFUSI is in control of ASSIZE and MEMLIMIT for the parent, the child's MEMLIMIT and ASSIZE will be set by IEFUSI. In both cases, if IEFUSI is active in the system and is attempting to control limits for an OMVS address space, the inheritance structure (Inhe) is ignored for MEMLIMIT and ASSIZE.

What happens if an identity change does not take place when a new process image is created by exec()?: When a new process image is created, limits are not affected unless a limit has been defined in the OMVS segment, an IEFUSI exit modifies the limit, or an identity change occurs. When the limit is controlled by system security or IEFUSI, the hard and soft limit are changed to the hard limit in the new image.

Specifying a new identity: You can specify a new identity in the following ways:

- Inheritance structure
- The _BPX_USERID environment variable
- pthread_security callable service
- _login
- setuid() function

For more information about APIs that affect process limits, see *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.

Note: If RACF or UNIX limits for CPUTIMEMAX, ASSIZEMAX, or MEMLIMIT are higher than the MVS limits, they are used for the hard limit.

Setting process limits in z/OS UNIX

System-wide limits are defined in the BPXPRMxx parmlib member. Table 4 lists the BPXPRMxx statements that you can use to set system-wide limits.

Table 4. System-wide limits that can be defined in BPXPRMxx

IPCMSGNIDS	IPCSHMMPAGES	MAXPTYS
IPCMSGQBYTES	IPCSHMNSEGS	MAXRTYS
IPCMSGQMNUM	IPCSHMSPAGES	MAXSHAREPAGES
IPCSEMNIDS	IPCSHMNIDS	MAXUIDS
IPCSEMNSEMS	MAXASSIZE	SHRLIBMAXPAGES
IPCSEMNOPS	MAXMMAPAREA	SHRLIBRGNSIZE

Process-level limits are defined in the BPXPRMxx parmlib member. Table 5 lists the BPXPRMxx statements that you can use to set process-level limits.

Table 5. Process-level limits that can be defined in BPXPRMxx

MAXCPUTIME	MAXFILEPROC	MAXQUEUEDSIGS
MAXPROCSYS	MAXPROCUSER	MAXTHREADS
MAXTHREADTASKS	MAXCORESIZE	MAXASSIZE

Table 6 lists the hard limits that can be defined in the RACF user profile.

Table 6. Hard limits that can be defined in the RACF user profile

ASSIZEMAX	The maximum address space size (RLIMIT_AS) for the z/OS UNIX user
CPUTIMEMAX	The maximum CPU time (RLIMIT_CPU) for the z/OS UNIX user
FILEPROCMAX	The maximum number of files per process for the z/OS UNIX user
MEMLIMIT	The maximum storage above the bar (non-shared memory size). MEMLIMIT is initially set in the SMFPRMxx parmlib member.
MMAPAREAMAX	The maximum memory map size for the z/OS UNIX user
PROCUSERMAX	The maximum number of processes per UID for the z/OS UNIX user
SHMEMMAX	The maximum shared memory size
THREADSMAX	The maximum number of threads per process for the z/OS UNIX user

Steps for setting process limits in z/OS UNIX: **Before you begin:** You need to understand how the MVS limits affect z/OS UNIX processes. For an explanation of soft and hard limits, see “What are hard limits?” on page 74 and “What are soft limits?” on page 74 You can modify some limits. Parameters are provided by batch JCL, TSO logon, and started tasks to limit region size and high memory. **Some MVS system-wide limits such as MEMLIMIT are initially set in SMFPRMxx.**

You also need to have planned the system-wide limits that will affect all z/OS UNIX users. For more information, see the section on defining system limits in the *Customizing z/OS UNIX* chapter.

Perform the following steps to set process limits in z/OS UNIX.

1. Determine the sources that your installation uses to control MVS limits and z/OS UNIX limits. For more details, see “Tuning limits in BPXPRMxx” on page 62. Also see the section on limiting the use of memory objects in *z/OS MVS Programming: Extended Addressability Guide*, which discusses region sizes.

2. Specify the z/OS UNIX system-wide limits in BPXPRMxx. The system-wide limits are listed in Table 4 on page 77. If you do not set them, defaults are used. To look up the defaults, go to *z/OS MVS Initialization and Tuning Reference*.

3. Specify higher limits for individual processes, if needed. For an explanation about setting process limits in general, see the section on setting limits for users in the chapter *Establishing UNIX security*. Also see the section on limiting the use of memory objects in *z/OS MVS Programming: Extended Addressability Guide*, which discusses the use of MEMLIMIT.

   You can use any of the following to change z/OS UNIX limits:

   - The SETOMVS operator command
   - z/OS UNIX programming APIs
   - RACF
   - z/OS UNIX commands
   - setrlimit
   - spawn()

   You can use the RACF ADDUSER and ALTUSER commands to specify the ASSIZEMAX limit on a per-user basis. Table 6 on page 77 lists the process limits that you can set.

   Rule: To define or change information in the OMVS segment of a user profile, including your own, you must have the SPECIAL attribute or at least UPDATE authority to the segment through field-level access checking.

   For more information about the OMVS segment in RACF user profiles and a complete list of what you can specify, see *z/OS Security Server RACF Security Administrator's Guide*.

   Resource values that processes receive when they are dubbed a process will use the RACF profile to determine the hard limit if it is higher than the soft (current) limit. It is also used when processes are initiated by a daemon process using an exec after setuid(). In this case, both the RLIMT_AS hard and soft limits are set to the address-space-size value

When you are done, you have set process limits in z/OS UNIX.

Note: Limits defined in the RACF user profile or modified by the IEFUSI installation exit override limits defined by z/OS UNIX processes.

Using the IEFUSI installation exit to set process limits: The IEFUSI and IEALIMIT installation exits control region size and memory above the bar, but z/OS UNIX ignores changes made by IEALIMIT. Changes made by IEFUSI to the region limit and hi memory limit are used if a RACF user profile is not used to define these values for the user. The IEFUSI exit can change the values used by the system for virtual storage available above and below the 16M line, and hi memory. Normally, z/OS UNIX takes action to ensure that these changes are honored. When a limit is specifically defined for a user via the OMVS segment, z/OS UNIX will ignore the

changes made by the IEFUSI exit if the ASSIZE or MEMLIMIT in the RACF OMVS segment is specified as a higher value than IEFUSI.

Because z/OS UNIX limits are normally inherited from MVS, the IEFUSI exit has already had a chance to modify the region size and MEMLIMIT for TSO, batch, and started task. The exit does not have to be called again during OMVS subsystem processing. z/OS UNIX processes such as telnet and rlogin do not have a chance to change the limits, so IEFUSI is not needed to control those limits.

Rule: If you install your own IEFUSI exit, update your SMFPRMxx parmlib member to exclude OMVS work and use z/OS UNIX to control the process limit. Specify:

```
SUBSYS(OMVS,NOEXITS)
```

Message IEE968I is issued when the SET SMF= command is processed because z/OS UNIX does not support the SSI Notify function. You can ignore this message.

After a hard limit is defined in the user RACF profile, the parents' hard and soft values for that limit will override IEFUSI, MVS, and z/OS UNIX changes in any children or executed programs. An executed or spawned process after an identity change always sets hard and soft limits to the RACF OMVS segment limit of the new identity, if the RACF OMVS segment limits are set for that identity. Other processes exhibit this behavior only if the value in the user RACF profile was higher than the value provided by MVS when the process was dubbed.

For more information about the IEFUSI installation exit, see:

- The section on IEFUSI in the *Managing account work* chapter
- *z/OS MVS Installation Exits*

Displaying process limits

You can use any of the following to display process limits:

- **ps** shell command
- **ulimit** shell command
- D OMVS,LIMIT operator command
- D OMVS,OPTIONS
- MEMLIMIT
- ASSIZEMAX

You can display the RACF limit using the following RACF command:

```
LU user NORACF OMVS
```

For more information about how to obtain security information for users, see the *Establishing UNIX security* chapter. You can obtain the information if the security administrator has set up field-level access for users for the OMVS segment of the RACF user profile as described in "Setting up field-level access for the OMVS segment of a user profile" on page 14.

Example:

```
ps
```

Result:

```
/shut/home/wellie2==>ps  ; alias to ps -o ruser,pid,vsz,vsz64,vszlmt64,comm
 RUSER  PID  VSZ VSZ64  VSZLMT64 COMMAND
   Erin  24 2432  0     20M sh -L
   Erin  29 2432  0     20M /tst/bin/ps -oruser,pid,vsz,vsz64,vszlmt64 -oargs
```

Example:

```
ulimit -a
```

Result:

```
core file         8192b
cpu time          unlimited
data size         unlimited
file size         unlimited
stack size        unlimited
file descriptors  256
address space     259192k
memory above bar  17592186040320m
```

Example: Assuming that the PID is 0050331651:

```
d omvs,limits,pid=0050331651
```

Result:

```
BPXO051I 16.42.32 DISPLAY OMVS 277
OMVS     000D ACTIVE          OMVS=(6F)
USER     JOBNAME  ASID        PID       PPID STATE   START     CT_SECS
MEGA     MEGA1    0020 0050331651   33554434 1CI--- 16.31.46      .051
  LATCHWAITPID=         0 CMD=sh -L
PROCESS LIMITS:        LIMMSG=NONE
                   CURRENT  HIGHWATER    PROCESS
                     USAGE      USAGE      LIMIT

MAXFILEPROC              6          7        256
MAXFILESIZE            ---        ---    NOLIMIT
MAXPROCUSER              3          4    NOLIMIT
MAXQUEUEDSIGS            0          1       1000
MAXTHREADS               0          0        200
MAXTHREADTASKS           0          0       1000
IPCSHMNSEGS              0          0        500
MAXCORESIZE            ---        ---    4194304
MAXMEMLIMIT              0          0     16383P
```

If you do not specify the PID, the display will show the system-wide limits

Changing process limits

You can set a system-wide limit using these BPXPRMxx statements and then specify the process limit using the RACF ADDUSER or ALTUSER command. The OMVS operator command can also be used.

- MAXASSIZE
- MAXCPUTIME
- MAXFILEPROC
- MAXMMAPAREA
- MAXPROCUSER
- MAXTHREADS
- MEMLIMIT

To change the MEMLIMIT of a specific process (or to be more exact, to change the MEMLIMIT value of the ASID that the process is running in), use the SETOMVS command.

Steps for changing the process limits for an active process: **Before you begin:** You need to know the PID of the process. To find the process, issue D OMVS,A=ALL.

Perform the following steps to change the process limits for an active process.

1. Display information about the current parmlib limits for a process with the ID *nnn*.

   Example:

   `d omvs,limits,pid=nnn`

   Result: You will get a display similar to the following:

```
BPXO051I 16.42.32 DISPLAY OMVS 277
OMVS     000D ACTIVE          OMVS=(6F)
USER     JOBNAME  ASID        PID       PPID STATE   START     CT_SECS
MEGA     MEGA1    0020 0050331651   33554434 1CI--- 16.31.46      .051
  LATCHWAITPID=         0 CMD=sh -L
PROCESS LIMITS:        LIMMSG=NONE
                   CURRENT  HIGHWATER    PROCESS
                     USAGE      USAGE      LIMIT

MAXFILEPROC              6          7        256
MAXFILESIZE            ---        ---    NOLIMIT
MAXPROCUSER              3          4    NOLIMIT
MAXQUEUEDSIGS            0          1       1000
MAXTHREADS               0          0        200
MAXTHREADTASKS           0          0       1000
IPCSHMNSEGS              0          0        500
MAXCORESIZE            ---        ---    4194304
MAXMEMLIMIT              0          0     16383P
```

---

2. Change the high memory limit, MEMLIMIT, in the address space containing PID *nnn*.

   Example:

   `SETOMVS PID=nnn, memlimit=16M`

   Result: You will get a message that the SETOMVS command was successful.

---

3. Check that the limit has been changed.

   Example:

   `D OMVS,LIMITS,PID=nnn`

   Result: You will see a display similar to the following:

```
BPXO051I 16.44.40 DISPLAY OMVS 283                                    C
OMVS     000D ACTIVE          OMVS=(6F)
USER     JOBNAME  ASID        PID       PPID STATE   START     CT_SECS
MEGA     MEGA1    0020 0050331651   33554434 1CI--- 16.31.46      .051
  LATCHWAITPID=         0 CMD=sh -L
PROCESS LIMITS:        LIMMSG=NONE
                    CURRENT  HIGHWATER    PROCESS
                      USAGE      USAGE      LIMIT

MAXFILEPROC             6          7        256
MAXFILESIZE           ---        ---    NOLIMIT
MAXPROCUSER             3          4    NOLIMIT
MAXQUEUEDSIGS           0          1       1000
MAXTHREADS              0          0        200
MAXTHREADTASKS          0          0       1000
IPCSHMNSEGS             0          0        500
MAXCORESIZE           ---        ---    4194304
MAXMEMLIMIT             0          0        16M *
```

When you are done, you have changed the user limit.

Reference information

For more information, see the following:

- The defining z/OS UNIX users to RACF section in the *Establishing UNIX security* chapter.
- The setting up default OMVS segments section in the *Establishing UNIX security* chapter.
- The setting limits for users section in the *Establishing UNIX security* chapter.
- The obtaining security information about users section in the *Establishing UNIX security* chapter.
- For information about the RACF ADDUSER and LISTUSER commands, see *z/OS Security Server RACF Command Language Reference*.
- *z/OS MVS System Management Facilities (SMF)*
- *z/OS MVS Initialization and Tuning Reference*
- *z/OS MVS Programming: Extended Addressability Guide*

Chapter 2. Changes for z/OS UNIX System Services Command Reference

ulimit — Set process limits

Format

ulimit [–**SHaAcdfMnst**] [*num*]

Description

ulimit sets or displays the resource limits on processes created by the user.

Options

–S Set or display the soft limit(s). The soft limit may be modified to any value that is less than or equal to the hard limit. For certain *resource* values, the soft limit cannot be set lower than the existing usage.

–H Set or display the hard limit(s). The hard limit may be lowered to any value that is greater than or equal to the soft limit. The hard limit can be raised only by a process which has superuser authority.

–a Display all resource limits that are available.

-A Set or display the maximum address space size for the process, in units of 1024 bytes. If the limit is exceeded, storage allocation requests and automatic stack growth will fail. An attempt to set the address space size limit lower than the current usage or higher than the existing hard limit will fail.

–c Set or display the core file limit. The core file limit is the maximum size of a dump of memory (in 512–byte blocks) allowed for the process. A value of 0 (zero) prevents file creation. Dump file creation will stop at this limit.

–d Set or display the data size limit. The data size limit is the maximum size of the break value for the process, in units of 1024 bytes. This resource always has unlimited hard and soft limits.

–f Set or display the file size limit. The file size limit is the maximum file size (in 512–byte blocks) allowed for the process. A value of 0 (zero) prevents file creation. If the size is exceeded, a SIGXFSZ signal is sent to the process. If the process is blocking, catching, or ignoring SIGXFSZ, continued attempts to increase the size of a file beyond the limit will fail.

–M Set or display the amount of storage above the 2 gigabyte bar that a process is allowed to have allocated and unhidden, in megabyte increments. An attempt to set the storage size limit lower than the current usage or higher than the existing hard limit will fail.

Tip: The amount of storage that **ulimit -M** displays does not necessarily reflect the MEMLIMIT setting found in the user's RACF OMVS segment. The value displayed will depend on how the user entered the OMVS shell and whether a change of identity was performed.

–n Set or display the file descriptors limit. The file descriptors limit is the maximum number of open file descriptors allowed for the process. This number is one greater than the maximum value that may be assigned to a newly created descriptor. Any function that attempts to create a new file

© Copyright IBM Corp. 1996, 2005

descriptor beyond the limit will fail. An attempt to set the open file descriptors limit lower than that already used will fail.

–s Set or display the stack size limit. The stack size limit is the maximum size of the stack for a process, in units of 1024 bytes. The stack is a per-thread resource that has unlimited hard and soft limits.

–t Set or display the cpu time limit. The cpu time limit is the maximum amount of CPU time (in seconds) allowed for the process. If the limit is exceeded, a SIGXCPU signal is sent to the process and the process is granted a small CPU time extension to allow for signal generation and delivery. If the extension is used up, the process is terminated with a SIGKILL signal. An attempt to set the CPU limit lower than that already used will fail.

num The new limit. *num* can be specified as “unlimited”.

Usage notes

1. **ulimit** is a built-in shell command.
2. If the command fails because of an attempt to set a resource limit lower than the current amount in use or higher than the existing hard limit, the resulting error message may indicate an invalid argument.

Localization

ulimit uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **NLSPATH**

Related Information

setrlimit in *z/OS XL C/C++ Run-Time Library Reference*.

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 such products 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 Vol 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:

`www.ibm.com/servers/eserver/zseries/zos/bkserv/`

© Copyright IBM Corp. 1996, 2005

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 products and/or the programs 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.

© Copyright IBM Corp. 1996, 2005

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:

DB2
IBM
Language Environment
MVS
OS/390
RACF
RMF
z/OS

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.

Program Number: 5694-A01, 5655-G52

Printed in USA