HEWLETT PACKARD

HP 9000 Computers

How HP-UX Works: Concepts for the System Administrator

How HP-UX Works: Concepts for the System Administrator

HP 9000 Computers

HEWLETT PACKARD

HP Part No. B2355-90029
Printed in USA August, 1992

Third Edition
E0892

Legal Notices

The information in this document is subject to change without notice.

Hewlett-Packard makes no warranty of any kind with regard to this manual, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Warranty. A copy of the specific warranty terms applicable to your Hewlett-Packard product and replacement parts can be obtained from your local Sales and Service Office.

HEWLETT-PACKARD COMPANY
3000 Hanover Street
Palo Alto, California 94304 U.S.A.

Printing History

This edition documents the 9.0 release of HP-UX, and is intended as a companion manual to the *System Administration Tasks* manual.

The printing date and part number indicate the manual's current edition. The printing date changes with each new edition. Minor changes may be made at reprint without changing the printing date. The manual part number changes when extensive editorial changes are made.

Manual updates may be issued between editions to correct errors or document product changes. To ensure that you receive the updated or new editions, you should subscribe to the appropriate product support service. See your HP sales representative for details.

- September 1989—First edition, as *HP-UX System Administration Concepts*
- January 1991—Second edition, as *How HP-UX Works: Concepts for the System Administrator*
- June 1991—Third edition
- August 1992—Fourth edition

This Edition documents the 9.0 release of HP-UX, and is intended as a companion manual to the *System Administration Tasks* manual.

Contents

Figures

Tables

1

Introduction

This manual supplements the *System Administration Tasks* manual to describe the essential concepts of system administration. Whereas the *System Administration Tasks* manual describes how to administer your HP-UX system, this manual discusses the underlying principles.

Use the *System Administration Tasks* manual to perform specific system administration tasks, such as adding a new user or reconfiguring operating-system parameters. Use this manual to learn about what happens to the operating system when you administer it (for example, how does the system get to a multi-user state during system startup; how does the virtual memory subsystem apportion user address space on each HP-UX platform).

This manual is especially aimed at users who simply have a need to know. Some users are satisfied with procedural explanations of how to do things; others require more background conceptual information. *How HP-UX Works* attempts to address the concerns of these users.

This introductory chapter contains the following:

- Typographical conventions used in this manual.
- Table of chapter contents.
- Description of the System Administration Manager (SAM).
- List of related system-administration manuals.

Typographical Conventions

Throughout this manual, the following typographical conventions are used:

If You See This ...	It Means ...
term	**Boldface** text indicates a term being introduced for the first time in a chapter.
`command`	`typewriter` text denotes commands, file names, and information displayed by the computer.
emphasis	*italic* text is used for emphasis (for example: *never* remove this file ...).
refpage(1)	is a reference to a page in the *HP-UX Reference*. This particular example says that the page *refpage* is found in section 1 of the *HP-UX Reference*.

<u>what you type</u>	<u>underlined</u> text is used in examples to distinguish what you type from what the computer displays; for example:
	`% ls`
	`reports personal projects`
	When the user typed `ls` to the HP-UX prompt (`%`), the system displayed three file names: `reports`, `personal`, `projects`.

Chapter Summaries

Chap No.	Title	Summary
2	System Startup	What happens from the time you restart your system to the time you get a `login:` prompt.
3	System Shutdown	Different methods of shutting down a system and what happens when you do so.
4	Login	What happens from the time you respond to the `login:` prompt to the time you get a shell prompt.
5	Process Management	What processes are, process states, commands for managing processes, and a discussion of multi-processing.
6	Run-Levels	What run-levels are and how they are set.
7	Memory Management	How HP-UX implements demand-paged virtual memory, basics of physical memory and swap-space management.
8	HFS File System	Overview of the file system characteristics, relationship to physical disks, file-system creation, storage, modification, and protection.
9	Logical Volume Manager (LVM)	Conceptual discussion of a Series 800 subsystem for managing disk space.
10	System Architectures	Survey of the various HP platforms for HP-UX—Series 300, 400, 700, and 800 platforms.
11	System Configuration	Descriptions of the HP-UX kernel, device drivers, and device files, and their combined role in system configuration.
12	HP-UX Peripherals	Information on tape and disk drives, CD-ROM file system, terminals, and modems.
13	Networking	Pointers to networking documentation.
14	System Accounting	Discussions of various approaches to account for computer usage.
A	Glossary	Definitions of all pertinent terms used in the manual.

The System Administration Manager (SAM)

The System Administration Manager (SAM) is a user interface for performing most routine administrative tasks without using the underlying HP-UX commands. For maximum flexibility, the *System Administration Tasks* manual documents both SAM and commands methods of performing these tasks.

SAM enables you to manage:

- Peripheral devices
- File Systems and disk space
- Routine tasks, such as backup, recovery, and user accounts
- Processes, including control, and monitoring
- Kernel and cluster configuration
- Network connectivity
- Remote administration
- Auditing and security

SAM in HP-UX 9.0 works on a new, object-oriented paradigm: You list or view information in a functional area and choose an action to perform your task.

To run SAM, type:

```
/usr/bin/sam
```

Activating the (Help) button from a dialog or message box gives you information about the attributes and tasks you can perform from the currently displayed window.

Choosing an item from the Help menu within a functional area gives you information about:

- the current functional area
- keyboard navigation within SAM
- using the SAM help system
- displaying the version of SAM you are currently running

Pressing the (f1) key gives you context-sensitive information for the object field at the location of the cursor.

See Chapter 1, "Introduction to System Administration", in the *System Administration Tasks* manual for detailed information about using SAM.

Related System-Administration Manuals

How HP-UX Works supplements information in the following procedural and reference manuals:

- *System Administration Tasks* manual.
  Provides detailed procedures for most system administration tasks.
- *HP-UX Reference.*
  Contains specifications of all HP-UX commands, system calls, library routines, and system files.
- *Installing and Updating HP-UX.*
  Describes how to install (or reinstall) HP-UX, and how to update HP-UX from a previous release.
- *Installing Peripherals.*
  Discusses how to install and configure peripheral devices on your HP-UX system.
- *Solving HP-UX Problems.*
  Provides guidance for troubleshooting problems in key areas of HP-UX.
- *Managing Clusters of HP 9000 Computers* (Series 300/400/700 only).
  Details how to set up, configure, and maintain computers belonging to an HP-UX cluster.

The following books will help you acquire basic HP-UX skills:

- *Using HP-UX*, HP part number B2910-90001.
- *HP Visual User Environment User's Guide*, HP part number B1171-90061.
- *The Ultimate Guide to the vi and ex Text Editors*, ISBN 0-8053-4460-8.

For pointers to networking documentation, refer to Chapter 13, "Networking Overview" of this manual.

2

System Startup

From the time you boot your system to the time you get a `login:` prompt, the system performs several important tasks automatically. The system tests computer hardware, loads and initializes HP-UX, communicates messages to users, and runs scheduled routines. When these system startup tasks are finished, HP-UX is in the **initdefault** run-level. As shipped, this is run-level 2 or multi-user HP-UX.

Note In case of trouble when starting up Series 300, 400, or 700 systems, you can boot from a recovery system (a minimal system kept on DDS media or cartridge tape). Be sure to make a recovery system as soon as you verify that the system is installed correctly. Refer to Chapter 2, "Constructing an HP-UX System" of the *System Administration Tasks* manual for procedure and *mkrs*(1M) of the *HP-UX Reference* for explanation of command syntax.

System startup occurs in two phases:

- The Boot ROM startup sequence.
- The HP-UX startup sequence.

As these phases occur, a series of messages appears quickly on the console screen. You can review the HP-UX messages once you have a `login` prompt by invoking the command `/etc/dmesg` with superuser privilege.

The Boot ROM startup sequence is different for each architecture, but the HP-UX startup sequence is virtually identical.

This chapter describes the Boot ROM startup sequence for Series 300/400, 700, and 800 computers, and the HP-UX startup sequence. By understanding them, you can tailor system startup to your unique needs.

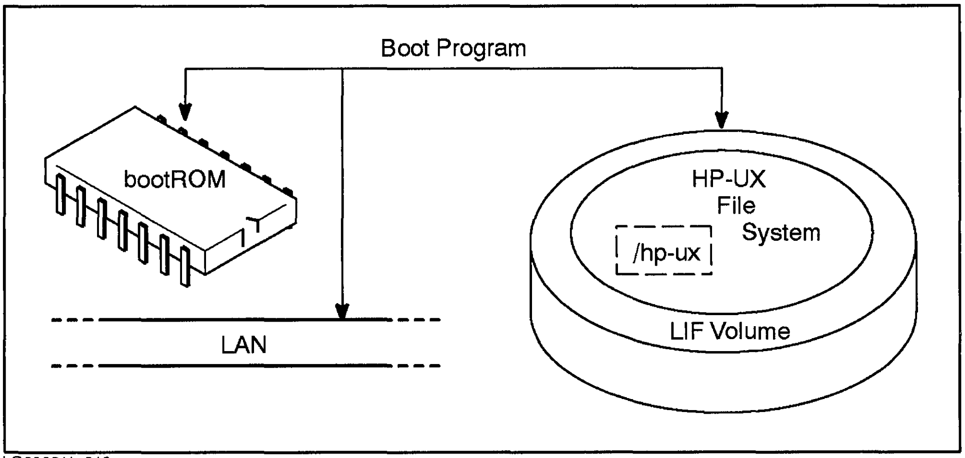
Boot ROM Startup Sequence Overview

Conceptually, the boot program performs the same basic functions regardless of architecture. When the system is powered up, the boot program initializes and tests hardware to bring the system into a usable state by the operating system.

This includes:

- test LEDs
- determine processor type
- initialize and test timers
- find system console and initialize video circuitry
- load configuration data from EEPROM
- reset and initialize I/O, including audio and HP-HIL
- display copyright and Boot ROM banners, CPU type, and EEPROM status
- test utility chip
- test RAM, report amount of memory found and any failures
- test DMA
- find and report built-in interface circuits
- test SCSI and LAN interfaces
- display boot choices

After checking the viability of the system, the boot program searches for a copy of the operating system (`/hp-ux`). It searches a list of potential sources, including disks and network (LAN), as shown by Figure 2-1. (LIF implementation differs, depending on architecture, but the basic concept remains true.)



LG200211_013

Figure 2-1. The Boot Program Searches for an Operating System

From the first available source it finds, the boot program also locates another program, called a secondary loader, and loads it into memory. The secondary loader then loads `/hp-ux` into memory and starts it up to enable you to use your system.

Secondary Loader

The boot program contains code to work with a specific hardware architecture. The secondary loader is a larger program, whose additional code provides the flexibility to deal with the changes to the booting process from operating-system release to release. (From the perspective of the boot program, the terms operating system and secondary loader are synonymous.)

Further, the secondary loader is written in LIF (Logical Interchange Format), code that is understood by HP 1000, 3000, and 9000 systems.

The boot program finds the secondary loader on the boot media (typically in the LIF volume of a mass storage media), loads it into memory, and starts it running.

Boot ROM Startup Sequence (Series 300/400)

The Boot ROM is a small machine-language program that resides in your computer's ROM (Read-Only Memory) on the SPU (System Processing Unit). When you boot (or power-up) the system, the computer starts the Boot ROM program, which causes the CPU to execute instructions that get the system running.

On Series 300/400 computers, the Boot ROM performs several steps during system startup:

- Locate and write to a system console.
- Search and test hardware internal to the SPU.
- Find, load, and start an operating system (in this case, HP-UX).
- Optionally, configure internal interfaces in the SPU.

Boot ROM Finds a System Console

The **system console** is the terminal (keyboard and display) to which the Boot ROM and HP-UX communicate with the system administrator. Boot ROM error messages, HP-UX system error messages, and certain system status messages appear on the system console screen. Sometimes (such as during the administrative run-level), the system administrator can communicate with HP-UX *only* through the system console.

The Boot ROM searches for a system console until it finds:

1. An RS-232C serial interface with its remote bit set.

   If the Boot ROM finds more than one serial interface card with its remote bit set, it selects the card with the lowest select code. In the case of the HP 98642A (four-channel multiplexer), port 1 is used.
2. An internal bit-mapped display. There can be only one internal bit-mapped display. Note that an HP 98700H display station can have its display interface card (HP 98287A or HP 98720) configured for internal or external control; if configured for external control, it is never chosen.
3. An external bit-mapped display with select code 132-255. They are selected in order of increasing select code.
4. An HP 98546A compatibility video interface.

5. An RS-232C serial interface *without* its remote bit set. If more than one is present, the one with the lowest select code is used. In the case of the HP 98642A (four-channel multiplexer), port 1 is used. The Boot ROM does *not* recognize the HP 98628 serial interface card as console when this condition is met; however, HP-UX does.

HP-UX communicates with the system console through the special device file, `/dev/console`. Although the boot ROM can function without a system console, HP-UX cannot. Therefore you must have a system console in one of the locations specified below to boot HP-UX.

Serial Interface Line Control Switch Pack Settings

When a serial interface card is selected as the system console, the boot ROM requires that the **line control switch pack** settings on the card be set to the same settings as the connected remote terminal. These settings provide the communications protocol for the console. Assuming an RS 232-C console, HP-UX resets the following values:

Stop Bits	1
Baud Rate	9600 bps
Parity/Data Bits	0's/7
Enq/Ack	NO
Pace (Handshake)	XON/XOFF

If the serial interface is part of the built-in Human Interface card, these values are set appropriately and you need not change them. Other configurations might require different line control switch pack settings on the interface card; see the documentation that comes with the card or the console for information.

Boot ROM Tests Hardware

The Boot ROM searches for and tests hardware internal to the SPU, including interface cards, RAM, and internal peripherals and chips. It displays names of installed peripherals and internal hardware. If hardware fails a test, the Boot ROM displays an error message. For example, two interfaces set to the same select code will cause a boot ROM error. For details on the tests performed, refer to your system's hardware *Installation Reference*.

Boot ROM Finds and Loads an Operating System

As explained earlier, much of the code in the Boot ROM consists of the **secondary loader**, which executes during system boot and is able to load the HP-UX system kernel, despite variations from release to release. Using the secondary loader, the Boot ROM searches for an operating system on connected mass storage media, or on cluster servers on your computer's local area network (LAN).

The Boot ROM has two modes for selecting an operating system: unattended and attended. In **unattended mode**, the Boot ROM automatically boots the first operating system found. Unattended mode is the default method for loading an operating system. In **attended mode**, you select the operating system to boot from all operating systems found.

If, for any reason, the Boot ROM is unable to find and load an operating system, it displays messages to that effect on the system console. These messages are described in the "Boot ROM Error Messages" section of your computer's *Installation Reference*.

Unattended Mode

Use the unattended mode of booting if you have only one bootable operating system online, or if you know the operating system you wish to boot is the first operating system the Boot ROM will find.

The Boot ROM searches for an operating system by a prioritized list of select codes. This includes LAN cards, disks, and other peripheral devices. The first operating system found on one of these devices is loaded. If no operating system is found, the list will be searched again until a system is found. This means that disks not found at power-up will be found after their initialization is complete.

To load HP-UX using unattended mode:

- Make sure that HP-UX is the first system found, following the prioritized list below.
- Make sure the device holding the HP-UX operating system is fully powered up and has achieved a ready state *before* booting the computer. If the disk drive has not completed its power-up sequence (which may require several minutes on some disk drives), the Boot ROM will not be able to access the

disk and load the system. If you are booting a cluster client, make sure the cluster server has completed the bootup sequence, including clustering and boot daemons.

Boot ROM Search Sequence.
The Boot ROM searches for an operating system in the following sequence:

1. SCSI disks at select codes 0-31, beginning from highest priority address (that is, 7), then searching addresses in decreasing numerical order (that is, 6, 5, and so on).
2. HP-IB disks at select codes 0-31, bus address 0, unit 0, volumes 0 through 7. (Note, however, that the volume scan differs on older Series 300 systems.)
3. Shared Resource Managers (SRM) at Select Code 21, Volume 8.
4. LAN interface at Select Code 21.

   If there are multiple remote servers at the same select code (as in an HP-UX cluster), the first server to respond is the first system listed on the menu (if attended mode) or the remote system selected (if unattended mode). This can change each time you boot your system.
5. HP 98259 (Bubble Memory) at Select Code 30.
6. HP 98255 (EPROM card) on Unit 0.
7. ROM systems (for ROM-based operating systems).
8. Remaining SCSI devices at Select Codes 0 through 31 bus addresses (device ID) 4-0, units 0-16. (note: the search order on SCSI addresses is high to low.)
9. Remaining HP-IB devices at select codes 0-31, bus addresses 0-7, units 0-16, volumes 0-7.
10. Remaining SRMs at Select Codes 0-31 (not at Select Code 21, Volume 8).
11. Remaining LANs (not at select code 21).
12. Remaining HP 98259 (Bubble Memory) on Select Code 0-29 and 31.
13. Remaining HP 98255 (EPROM) units.

Each category in the search sequence might have multiple units at the same select code and bus address, all of which are searched before moving to the next ascending select code or bus address. Thus, an HP-UX system on the root mass storage device at select code 14, bus address 0, unit 0, is found and loaded before any operating systems at select code 14, bus address 0, unit 1.

Boot Search Criteria.
The boot search sequence given is accurate for Revision 2.0 Boot ROM. Different revisions of the Boot ROM might vary slightly. Regardless of revision, however, the boot search sequence was developed on the following guidelines:

1. High-to-low priority addresses
2. Operating systems external to the SPU before ROM- or EEPROM-embedded operating systems
3. Local systems (dedicated devices) before remote systems (shared devices, such as a LAN server)
4. Removable media before non-removable media
5. Internal systems before external systems (Internal refers to operating systems on integrated devices)
6. Small-capacity devices before large-capacity devices

Other points worth noting:

- You must have Boot ROM rev. C or later to boot from a SCSI disk drive. At rev. C, SCSI disk drives are treated the same as any other disk drive.
- You must have Boot ROM rev. B or later to boot across a local area network (LAN), as in the HP-UX clustered environment.

Series 400 Unattended-Mode Boot Variation.

Series 400 systems provide additional flexibility during unattended mode. If the Series 400 has never been booted, a Configuration Control menu appears, which allows you to:

- Boot automatically from the first operating system found, or
- Specify an operating system by using Auto System Selection.

```
        Configuration Control

   Keys  Mode Class
   -------------------------------------
   1  I/O Configuration
   2  Auto System Selection
   3  Boot Mode Selection

   A  Abort without changes
   -------------------------------------
      Type [key] RETURN ?
```

The I/O Configuration key is explained later in this chapter, since it applies to some Series 300 systems, as well as all Series 400 systems.

You can use the Auto System Selection key to override the Boot ROM search priority and specify which system to boot. Auto System Selection resembles Attended mode booting, but here, the Boot ROM will remember your choice for future use. You can change your system selection by re-entering this menu at a later time.

The Boot Mode Selection key allows you to choose either HP-UX or DOMAIN operating system. (DOMAIN is available on the Series 400 only.)

Attended Mode

If you want to choose which specific version of the operating system to be used by the Boot ROM, you must enter the attended mode of selection. For example, when you are installing HP-UX, you need to boot from the install media, so you would use attended mode.

To enter attended mode, type a space, Return, or any letter or number during the time:

- After the keyboard has been initialized (the keyboard is initialized when the word **Keyboard** appears on the system console)
- Before the Boot ROM finds a default system.

The character used to enter attended mode is used as part of the string to select the operating system.

The best way to enter the attended mode is to hold down the space bar until the word **Keyboard** appears in the installed interfaces list on the left side of the screen.

Once you enter attended mode, the Boot ROM displays a list of one or more loadable operating systems, along with their mass storage device name and address. The following HP-UX operating systems are possible:

SYSHPUX	contains a pointer to the file, **/hp-ux**, which is the kernel, or compiled core program of the HP-UX operating system.
SYSDEBUG	used only for device driver writing.
SYSBCKUP	contains a pointer to the file /SYSBCKUP, a backup kernel.

In unattended mode, the Boot ROM loads SYSHPUX as the HP-UX operating system.

If you are booting a cluster client, the cluster server (or servers) will look like this:

:LAN, 21, *cluster_server_name*

To select one of the operating systems listed, enter the two characters to the left of the operating system name and press (Return). Do *not* use the (Shift) key as the characters you type are automatically shifted to upper case. For example, if the Boot ROM displays this list:

```
:LAN, 21, your_system_name
   1H  SYSHPUX
   1D  SYSDEBUG
   1B  SYSBCKUP
```

to select SYSHPUX, you type (1)(H), and 1H appears at the lower-right corner of the screen.

For details on using attended mode, see the *System Administration Tasks Manual* and appendices of *Installing Peripherals Manual*.

Boot ROM I/O Configuration

The Boot ROM of Models 345, 375, and all Series 400 systems allow you to change the characteristics of some built-in I/O interfaces.

To select the I/O Interface Configuration menu, enter the key sequence (Ctrl-C)(Return), instead of using the space bar, to enter attended mode.

The I/O Interface Configuration menu displays a list of configurable interfaces (depending on which interfaces are installed in your SPU). When you make a selection, a menu specific to the chosen interface appears with appropriate configuration choices.

Depending on interface, configuration choices include the following kinds of settings:

Select Code	Set value for most interfaces. (Select code for standard HP-IB cannot be reconfigured.)
Interrupt Level	Set value for most interfaces.
Remote/Local	Convert a serial terminal to system console.
Fast/Normal	Set data transfer or handshake speed.
Modem	Enable modem handshaking for serial ports.
Parity	Check parity of data for SCSI transfers.
Bus Address	Set value for SCSI controller bus address.
HP-IB System Controller	Enable or disable.
DMA Mode	Set the width of data transfers.

Once made, the configuration values are stored in EEPROM, reprogrammable Electrically Eraseable Read-Only Memory. Like the Boot ROM, the contents of EEPROM are retained even if power is removed from the SPU.

Note that not all settings apply to every interface. For details on reconfiguring your internal interfaces, consult *BootROM Configuration Mode User's Manual* and *Installing Peripherals Manual*.

HP-UX Takes Control

Once the operating system is loaded, the Boot ROM passes program control to the operating system. The operating system then controls the system until you re-boot the system. A later section in this chapter, entitled "HP-UX Startup Sequence," describes what HP-UX does between the time it takes control and the time you see the `login:` prompt.

Boot ROM Startup Sequence (Series 700)

The automatic boot process on the Series 700 resembles the Series 800 moreso than the Series 300 or 400, due to similarities in the Series 700 and 800 processor-dependent code (PDC).

Switching on the system or pressing the system transfer-of-control [TOC] button causes PDC and I/O-dependent code (IODC) firmware to be executed, to verify hardware and system integrity. The PDC runs self-tests and locates the console, using IODC and the paths stored in Stable Storage. The PDC displays on the console screen copyright information, PDC and IODC ROM revisions, and amount of memory configured. (See the manual page for *pdc*(1M) in the *HP-UX Reference*.)

Once checking is complete, one of two things happen:

- If `AUTOSEARCH` is off, PDC searches for a potential boot device, as described in the section, "Attended Mode," which follows.
- If `AUTOSEARCH` is on, PDC boots from Stable Storage, or searches for an appropriate path for console and boot, if any paths in Stable Storage fail.

PDC displays the following to the console:

```
Selecting a system to boot.
To stop selection process, press and hold the Escape key.
```

If the [Escape] key is not pressed, PDC loads the initial system loader (ISL) from the primary path in Stable Storage (for example, a SCSI disk with address `scsi.6.0`) and transfers control to it.

The ISL, scheduled by the `autoboot` sequence, finds the `autoexecute` file and executes the command specified in it (usually `hpux` or `hpux boot disc(;0)/hp-ux`). By default, the `autoexecute` arguments load `/hp-ux` into memory and execute.

While loading, the secondary loader displays information about the device and file being booted, the text, data, and BSS size of the kernel and the kernel's startup address.

If necessary, you have 10 seconds to override the default `autoboot` sequence by pressing the [Escape] key and enter Attended Mode.

For details on the Series 700 boot capabilities, including boot sequences from a variety of devices, `autoboot`, and restore, see the manual page *hpux_700*(1M) in the *HP-UX Reference*. Also, see the *Owner's Guide* for your Series 700 system for information on secondary loader enhancements.

Attended Mode

Each time the computer is powered on, you have the opportunity to interact with it by entering Attended Mode. For example, you might need to interrupt the boot sequence for either of the following reasons:

- To redirect the boot sequence.
- To perform a boot administration function provided by the Boot Console User Interface.

Pressing the (Escape) key halts the automatic boot sequence and puts you into Attended Mode. The system searches the SCSI, LAN, and EISA interfaces for all potential boot devices—devices for which boot I/O code (IODC) exists. The system then displays a table, such as the following:

```
Device Selection     Device Path           Device Type
----------------     -----------           -----------
P0                   scsi.6.0              CD-ROM HEWLETT-PACKARD
P1                   lan.123456-789abc     hpwgux42
P2                   eisa.1.0.0.0.0.0      SCSI Disk Drive
```

At this point, the Boot Console User Interface main menu offers the following options:

```
b)   Boot from specified device
s)   Search for bootable devices
a)   Enter Boot Administration mode
x)   Exit and continue boot sequence
?)   Help
```

Using the `b)` option, you can direct `hpux` to boot from a specific device. For example, to direct `hpux` to boot from the SCSI disk drive, you would type the key sequence (P)(2).

Using the `s)` option of the Boot Console User Interface main menu, you direct the system to search through the list of potential boot devices for only those that have a LIF volume and initial program loader (IPL). The system then displays a table listing only those devices from which you can boot.

For example, a system might return the following selection:

```
Device Selection      Device Path           Device Type & Utilities
----------------      -----------           -----------------------

P0                    scsi.6.0              Quantum PD210S
                                              IPL
P1                    scsi.5.0              Quantum PD210S
                                              IPL
P2                    scsi.1.0              HP 2213A
                                              IPL
```

Boot Administration Mode

Using the a) option of the Boot Console User Interface main menu, you can enter Boot Administration mode, from which you can alter default behaviors exhibited at boot-up or obtain useful information about the hardware as configured. The following commands are available at the Boot Administration Mode menu:

```
AUTO            Display state of Autoboot/Autosearch flags
AUTOSEARCH      Set state of Autosearch flag
AUTOBOOT        Set stat of Autoboot flag
BOOT            Boot from Primary/Alternate path or Specified Device
DATE            Read/Set the Real-Time Clock
EXIT            Return to previous menu
FASTSIZE        Display/Set FASTSIZE memory parameter
HELP <item>     Display Help information for <item>
INFO            Display boot/revision information
LAN_ADDR        Display LAN Station Address
OS              Display/Select Operating System
PATH            Display/Modify Path Information
PIM_INFO        Display Processor Internal Memory Information
RESET           Reset the System
SEARCH          Search for boot device
SECURE          Display/set secure boot mode
SHOW            Display the results of the previous search
```

For detailed information on using the Boot Administration mode, see the *Owner's Guide for HP-UX Users* for your Series 700 system.

Boot ROM Startup Sequence (Series 800)

When more than one operating system is present on the system's mass storage devices, both the location of the operating systems and the type of media on which they are stored determine which operating system is loaded. The primary boot path in Stable Storage determines the default boot path. Stable Storage is the memory in Series 800 computers reserved for maintaining critical configuration parameters used during system boot. For example, the primary and alternate boot paths, console path, and autoboot settings are stored in Stable Storage.

The Boot ROM contains general-purpose software developed to support both present and future Hewlett-Packard operating systems. This section describes how the ROM boots the HP-UX operating system.

When you turn the computer on, the Boot ROM goes through the following sequence:

1. Performs System Processing Unit (SPU) self-test.
2. Reads the console path from Stable Storage, tests it, and assigns a display terminal for use as a system console.
3. Reads the boot device path from Stable Storage, searches for the paths on Model 825, 835, and 845 computers using a capability called "autosearch", or lets you enter the path from the system console and tests the boot device path.
4. Loads the Initial System Loader (ISL) into memory.

The media on which your bootable system resides has a boot area and a root partition. The boot area contains the bootstrap program and other files needed for bringing up the system. See *isl*(1M) in the *HP-UX Reference* for more information on these files. The disk section where the boot area resides is `/dev/dsk/c0d0s6`. The root partition contains a file, called `hp-ux`, which is the operating system. The typical disk section for the root partition on a system using an HP-IB interface card is `/dev/dsk/c0d0s0`.

System Boot Overview

At powerup or reset:

1. The processor selftest is executed from PDC (Processor Dependent Code).
2. The PDC locates the system console and tests the console path using I/O-Dependent Code. The boot and console paths are kept in Stable Storage. Stable Storage memory is static RAM.
3. The PDC checks to see if the autoboot flag is set. If not, the system asks for a boot path. If set, the system prompts the console with the following message:

   ```
   Booting from default lpath
   To interrupt, press any key within 10 seconds
   ```

   If you press a key the system asks if booting will be done from the primary autoboot path, the secondary autoboot path, or ask for an alternate path.
4. The PDC loads the ISL (Initial System Load). The ISL loads an operating system (HP-UX in this case).

Once the operating system is loaded, the Boot ROM passes program control to the operating system. The operating system then controls the system until you re-boot the system. The section in this chapter called "HP-UX Startup Sequence" describes what HP-UX does between the time it takes control and the time you see the `login:` prompt.

Boot ROM Search Sequence (Series 800)

The Boot ROM initializes the primary boot path, loads ISL, and allows you to select either the manual or autoboot mode. On Model 825, 835, and 845 computers, an additional mode called **autosearch** is available when autoboot is enabled. In manual mode, you can select the boot device from all the available peripheral devices. In autoboot mode, the Boot ROM automatically boots the operating system from the primary boot path defined in Stable Storage.

Autoboot

You should use **autoboot** except for first-time installation and operating system reconfiguration. The ISL `autoboot on` command enables autoboot. No reboot or automatic reboot on panic is possible. Panic is a condition when the system becomes inoperative due to an abnormal condition detected by the kernel. Before using autoboot, make sure the boot device is fully powered up and ready for operation before you turn on your computer. Autoboot is selected if you let the 10 second override period expire.

Manual Boot

Manual boot can be entered by pressing any key during the 10-second override period in the beginning of the autoboot sequence. When manual mode is activated, the Boot ROM prompts for the path to be used. The primary boot path is not altered or disabled.

If you do not want to boot automatically from the primary boot path during the boot process, disable the autoboot flag in Stable Storage. You can do this using the ISL `autoboot off` command. However, this is not normally done because the autoboot feature makes your system administration tasks more efficient. Disabling autoboot requires your intervention each time the system is rebooted. To re-enable autoboot, use the ISL `autoboot on` command.

Autosearch

Many Series 800 computers have a feature called **autosearch.** If the system cannot locate the console using the console path from Stable Storage, it searches for a console device. Then, if the system cannot locate the boot device using the primary or alternate boot paths and the autosearch flag is set, the system continues to search for a boot device.

The autosearch flag is much like the autoboot flag. It is in Stable Storage and can be enabled or disabled. Use the ISL **`autosearch on`** command to enable autosearch and the ISL **`autosearch off`** command to disable the feature.

When autosearch is invoked (if the two paths specified in Stable Storage fail), the following messages appear on the console:

```
Autosearch for boot device enabled.
To override, press any key within 10 seconds.
```

If you press a key, the system responds:

```
Do you want to continue an interactive search? (Y or N)?>
```

If you answer "no," autosearch halts at that point and proceeds to manual boot. If you answer "yes," the system searches for a boot path, states it, and asks you if you want to boot from it. If you respond "yes," the system uses that path. Otherwise, it presents other logical paths until you respond positively or until it finds no other paths. It then proceeds to manual boot.

HP-UX Startup Sequence

Once HP-UX takes control from the Boot ROM, it performs two tasks:

- Find the root file system.
- Start the `init` process and bring the system to run-level 2, which is multi-user HP-UX.

HP-UX Finds the Root File System

Once HP-UX starts, it searches for the root file system. The **root file system** is the portion of the file system that forms the base of the file system hierarchy—that is, the portion of the file system on which other file systems can be mounted. The root file system contains the critical files (such as the kernel in `/hp-ux`). Generally, the root file system is found on the disk from which HP-UX booted. For cluster clients, the root file system is on the cluster server. (Note, clusters are not supported on Series 800 as of HP-UX 9.0.)

After finding the root file system, the operating system starts a shell to read commands from `/etc/pre_init_rc`. Among these commands is *fsck*(1M), which checks the root file system (currently in a read-only state). (`fsck` is discussed in Chapter 8, "HFS File System" of this manual and in Chapter 6, "File System Problems", in *Solving HP-UX Problems*.) Any problems *fsck* encounters that it is unable to fix are caught later by `/etc/bcheckrc`. After *fsck* exits, the operating system remounts the file system in a read-write state.

Caution Do not modify the `/etc/pre_init_rc` script; it might cause the system to be unbootable.

HP-UX Starts the init Process

Then, HP-UX starts its first process, `/etc/init`. The `init` process has process ID one (1) and no parent process.

The `init` process reads the `/etc/inittab` initialization file to define the environment for normal working conditions.

System Initialization File—/etc/inittab

The `init` process reads the `/etc/inittab` file one line at a time, each line containing an entry that describes an action to take.

The syntax of `inittab` entries is:

id: *run-levels*: *action*: *process*

id A one- or two-character ID that uniquely identifies the entry.

run-levels Defines the run-levels in which the entry can be processed. You can specify multiple run-levels. If this field is empty, *all* run-levels are assumed.

Typically, entries tell `init` to run a process at specific run-levels. If no run-levels are specified, the process can execute in any run-level. For example, the following entry tells `init` to run the `/etc/getty` process in all run-levels:

```
cons::respawn:/etc/getty console H
```

action Identifies what action to take for this entry. The *actions* are as follows:

`sysinit` Performs system initialization on devices needed by `init` for obtaining run-level information at the console, such as `tty` characteristics. `sysinit` entries must finish executing before `/etc/inittab` continues.

`initdefault` Causes the initial (default) run-level to be the value of the *run-levels* field. If more than one run-level is specified in *run-levels*, `init` uses the highest specified run-level.

`boot` Run the command specified in the *process* field at boot-time only. Do not wait for *process* to die before reading the next entry.

Before enabling other users to access the system, `init` executes all `/etc/inittab` entries marked `boot` or `bootwait`. These processes are known as **boot processes**.

`bootwait` Run the command specified in the *process* field at boot-time only. Wait for *process* to die before reading the next entry.

The following entries tell `init` to execute the `recoversl` program and `brc` shell script, respectively. `init` must wait for each process to die before moving to the next entry.

```
slib::bootwait:/etc/recoversl </dev/console >/dev/console 2>&1 #shared libs
brc2::bootwait:/etc/brc >/dev/console 2>&1   #boottime commands
```

`wait` On entering the run-level that matches the *run-levels* field of this entry, run *process* and wait for it to die before reading the next entry.

`respawn` On entering the run-level that matches the *run-levels* field of this entry, run *process* if it is not already running. Do not wait for *process* to die before reading the next entry. When *process* dies, run it again.

process This is a shell command to be run, if the entry's *run-levels* matches the run-level and/or the *action* field indicates such action.

HP-UX recognizes any text following the `#` symbol as commentary.

Each system architecture has its own `/etc/inittab` file, with some unique architecture-specific `/etc/inittab` characteristics. For example, on the Series 800, first `/etc/inittab` calls `ioinit`, which initializes kernel I/O data structures using information from `/etc/ioconfig` and then calls `insf` to assign logical unit numbers and create special files for all new devices on the system. The `/etc/inittab` entry is as follows:

```
io::sysinit:/etc/ioinit -i >/dev/console 2>&1
```

Series 300/400/700 systems do not require this initialization of kernel I/O data structures.

Default Run Levels—initdefault

The `/etc/inittab` file sets up system run levels. The entry marked `initdefault` sets the default run-level to 2:

```
init:2:initdefault:
```

(See Chapter 6, "Run-Levels" for information about run levels.)

Among the `/etc/inittab` entries, are programs that ensure system integrity and set up essential processes. These programs are discussed in the next subheads.

Boot Check Run Command—/etc/bcheckrc

If your system is using disk mirroring (an optional feature of the Series 800 only), the `/etc/bcheckrc` (Boot Check Run Command) program calls `/etc/mirrorrc` as its first operation. `mirrorrc` configures mirrored disks, runs `fsck`, sets up logging for the mirrored disks, and reimages the disks.

If you are implementing the Logical Volume Manager (LVM), `bcheckrc` then calls `/etc/lvmrc` to activate LVM volume groups.

On the Series 700, if appropriate, `bcheckrc` runs `eisa_config`.

On all systems, `/etc/bcheckrc` verifies that the system was properly shut down. To determine this, `bcheckrc` calls the `fsclean` program, which checks each file system of type `hfs` in `/etc/checklist` for consistency. `fsclean` looks at a flag called the **clean byte** in the primary superblock of each file system. When a file system is created, the clean byte flag is set to `FS_CLEAN`. When the file system is mounted (using the `mount` command), the clean byte flag is set to `FS_OK`. During a normal shutdown (that is, during execution of the `reboot` or `shutdown` command), the clean byte is reset to `FS_CLEAN`. Thus, under normal conditions, the file system can be unmounted and set to `FS_CLEAN`, or mounted and set to `FS_OK`.

If it encounters the file system unmounted and its clean byte set to `FS_OK`, `fsclean` perceives the file system to be in an inconsistent state (due to a crash or other incorrect shutdown). In this case, `fsclean` exits and `bcheckrc` runs `/etc/fsck -P`, `fsck` in `preen` mode. This will correct most errors found.

If `fsck` run from `bcheckrc` fails for any reason, `bcheckrc` starts a shell with the prompt `(in bcheckrc)#`, instructing you to run `fsck` interactively. If this occurs, you *must* run `fsck` to ensure the integrity of your file system.

Some file system problems must be fixed in this way to minimize risk of data loss. After running `fsck` interactively, you may be instructed to reboot the system. If so, *reboot the system using* `reboot -n` *to ensure your system's* *integrity.* If `fsck` does not tell you to reboot, exit the shell by typing CTRL D. This returns control to `bcheckrc`.

`bcheckrc` then ensures that `/lib` is not a mounted volume. Since HP-UX uses shared libraries, `/lib` must be on the root partition, and thus available to all commands that need them to boot the system.

Finally, `bcheckrc` mounts all type `hfs` file systems and exits.

Shared Libraries Check—/etc/recoversl

After `bcheckrc` exits, `init` calls `recoversl` to check for the existence of shared libraries critical to the system. `recoversl` checks for (and corrects, if necessary) the owner, group, and permissions of these shared libraries.

The shared libraries considered critical to the system are `dld.sl` (the dynamic loader) and `libc.sl`. If any of these libraries are missing or damaged, `recoversl` guides the system administrator through procedures to recover the shared libraries from update media.

Additional Boot Tasks—/etc/brc

The `/etc/brc` program does these tasks:

- Sets the system PATH variable, which defines the default command search path for all users.
- Downloads the floating point microcode if your system is set up for the floating point accelerator. (Series 300 only)
- Removes the `/etc/rcflag` file (used later as a check for startup condition).
- Looks for and removes the `/etc/mnttab` file on a standalone or cluster server system. The `/etc/mnttab` file, a list of mounted file systems, is recreated later when `/etc/rc` remounts the file systems.
- Creates a new `/dev/crt` file, if needed. (Series 300 only.)

Initial Customization Script—/etc/rc

The following entry in `/etc/inittab` invokes `/etc/rc`:

```
rc::wait:/etc/rc >/dev/console 2>&1 # system initialization
```

The /etc/rc script consists of a main script program and several shell functions (subroutines in a shell program).

The main program checks to see if it is boot time according to whether the /etc/rcflag is present or absent. (If the boot process has proceeded properly, /etc/brc will have removed /etc/rcflag.) /etc/rc determines whether to perform system initialization and start various **daemon** (background) processes. The actual tasks performed by /etc/rc depend on the configuration of the machine (see Table 2-1). /etc/rc determines if you are running as a standalone system, a cluster server, or a cluster client. It then calls shell functions applicable to your system.

When finished, /etc/rc calls a shell function called localrc. This is the shell function you should use to customize the /etc/rc script. It should contain any tasks you wish to perform that are not part of the standard /etc/rc functions. In the localrc shell function, you can add commands you wish to perform every time the system is booted or whenever there is a change in run-level which init does not handle.

For each system state (standalone, cluster server, or cluster client), Table 2-1 shows what functions /etc/rc does. Note that some commands (such as networking commands) may not be available or installed. /etc/rc checks for the existence of all such commands before attempting to run them.

The init process waits until /etc/rc dies before processing the next entry in /etc/inittab.

Table 2-1. /etc/rc functions

Function	Stand-alone	Root Server	Diskless Cnode[1]
Mount all **hfs** volumes listed in the /etc/checklist file.	X	X	X
Initialization: set TZ and other variables; set device files used by /etc/rbootd, set up **vt** gateway.	X	X	X
Set host name in variable SYSTEM_NAME.	X	X	X
Set the date.	X	X	
Save a core image of a previously crashed system.	X	X	X
Start swapping to all swap devices in /etc/checklist	X	X	X
Start the syncer.	X	X	
Start the **lp** scheduler.	X	X	
Clean up editor and **uucp** files.	X	X	
Start networking.	X	X	X
Start cluster server processes (/etc/csp).		X	X
Start the remote boot daemon (/etc/rbootd).		X	X
Start **cron**.	X	X	X
Start **pty** allocation daemon.	X	X	X
Start **vtdaemon**.	X	X	X
List files found in /tmp and /usr/tmp directories.	X	X	X
Clean up logging (/usr/adm/$LOG) files.	X	X	X
Start diagnostic logging for the I/O subsystem (Series 700/800 only).	X	X	X
Start logging system messages (Series 700/800 only).	X	X	X
Start auditing processes.	X	X	
Set local initialization functions to **rc**.	X	X	X

1 HP-UX clusters are not supported on Series 800 as of HP-UX 9.0.

Powerfail Routines—/etc/powerfail

A local powerfail is a power failure that halts the computer by affecting its central bus. The Series 800 HP-UX operating system provides a mechanism for recovery from a local powerfail, to ensure that any program running on the system at the time of failure can resume executing when power to the bus is restored.

Powerfail routines are invoked from `/etc/inittab` to ensure that power is maintained to the system in case of emergency.

```
pf::powerwait:/etc/powerfail >/dev/console 2>&1 #power fail routines
```

If need be, powerfail can be disabled by reconfiguring an operating-system parameter as follows:

```
pfail_enabled 0;
```

Be sure to follow guidelines for correct shutdown and start-up of a system necessitated by powerfail. These guidelines are given in Chapter 3, "Starting and Stopping HP-UX" of the *System Administration Tasks* manual.

Note, although powerfail appears in all `/etc/inittab` files, the entry is only used on systems that support powerfail.

Terminal Processes Startup—/etc/getty

Once `/etc/rc` has finished its run-level 2 execution, control returns to `init`, which runs the commands from the *process* field of all run-level 2 entries in `/etc/inittab`. Typically, `/etc/inittab`'s run-level 2 command field entries consist of `/etc/getty` commands, one for each terminal on which users log in. (When you add a new terminal with the SAM utility, it automatically adds an appropriate `/etc/getty` entry to `/etc/inittab`.) The `/etc/getty` command runs the `login` process on the specified terminal, allowing users to login on the terminal.

For example, the following `/etc/inittab` entry runs a `getty` at the system console:

```
cons:0123456:respawn:/etc/getty -h console console  # system console
```

The `respawn` *action* field tells `init` to restart the `getty` process after it dies. This means that each time you log off the system console, a new `login:` prompt is displayed, so you can log in the next time. The `0123456` *run-levels* field indicates that `init` runs `getty` in run-levels 0 through 6. However, when `getty` respawns other terminal `login` processes, it is often set up to run only in run-levels 2 and 3 or only 2.

As shipped, `/etc/inittab` invokes `/etc/getty` only for the system console. If your system has additional terminals on which you wish to support logins, you must add the appropriate `getty` entries to `/etc/inittab`. (Note, however, that SAM automatically creates these entries when you use it to add terminals).

The `/etc/getty` command is the first command executed for each login terminal. It specifies the location of the terminal and its default communication protocol, as defined in the `/etc/gettydefs` file. It prints the `/etc/issue` file (if present) and it causes the first `login:` prompt to be displayed. Eventually, the `getty` process is replaced by your shell's process (see Chapter 4, "Login").

When you logout, the `/etc/init` process is signaled and takes control again. The `init` process then checks `/etc/inittab` to see if the process that signaled it is flagged as continuous (denoted by `respawn`). If the process is continually respawned, `init` again invokes the command in the command field of the appropriate `inittab` entry as described above (that is, the `getty` runs and a new `login:` prompt appears). If the process is not flagged as continuous, it is not restarted.

At the end of `/etc/inittab` is an entry that runs powerfail recovery procedures in case of power failure.

Note

Do *not* add `/etc/getty` entries to `/etc/inittab` for unconfigured terminals, unless action is "off".

If the system finds itself having to respawn entries too rapidly, it assumes that a problem exists and goes to sleep for five minutes before trying to respawn again. If the problem involves the `getty` for the system console, the system might not be bootable without repair.

Users can log in at all terminals for which `getty` processes have been executed.

For More Information

The sections in this chapter describe the default operation of the system as shipped to you. However, by altering certain configuration or system files, any of the procedures can change. If, for example, you write your own `/etc/rc` script, the paragraphs which follow may no longer apply.

Table 2-2 shows where to look for additional information.

Table 2-2. Additional Startup Information

To Learn More about ...	Refer to ...
Processes	Chapter 5, "Process Management"
Run-Levels	Chapter 6, "Run Levels"
`/etc/init`	Chapter 6, "Run Levels" *init*(1M) in the *HP-UX Reference*
`/etc/inittab`	Chapter 6, "Run Levels" *inittab*(4) in the *HP-UX Reference*
`/etc/brc`	the `/etc/brc` file *brc*(1M) in the *HP-UX Reference*
`/etc/bcheckrc`	the `/etc/bcheckrc` file
`/etc/rc`	the `/etc/rc` file
`/etc/getty`	*getty*(1M) in the *HP-UX Reference*
`/etc/recoversl`	*recoversl*(1M) in the *HP-UX Reference*

3

System Shutdown

System shutdown is the process of taking your system from a running state (either run-level 2 or run-level s) to a halted state in which no processes are running. This chapter discusses:

- Halting the system versus rebooting the system.
- Shutdown with the `shutdown` command versus the `reboot` command.
- What happens during shutdown.

Halting vs Rebooting

There are two levels of shutdown: halting and rebooting. **Halting** brings the system to a complete stop; in this state, the only way to restart the system is to cycle power or reset the hardware. **Rebooting** brings the system to a complete stop, but restarts the system as if you had booted it.

Whether to halt or reboot your system depends on why you want to shutdown in the first place: If you want to shut the computer off for an extended period of time (for example, to add new hardware or to leave the system off for a long weekend, etc), then halting is appropriate. If you want to shutdown only to boot the system again (for example, to use a newly reconfigured kernel), then rebooting is appropriate.

Shutdown vs Reboot

After deciding whether to halt or reboot the system, you must decide which command to use: `shutdown` or `reboot`. Which command you use depends on:

3

- Whether users are logged in
- How quickly you need to shut the system down.

When to Use shutdown

The `shutdown` command shuts down more slowly than `reboot`, but more gracefully. It uses `kill -14` to kill running processes, which lets processes terminate naturally within a grace period. This is the safest way to shutdown and it helps ensure file system integrity.

It also displays messages directing users to log off within a specified grace period. You can specify the grace period when you invoke `shutdown`. It is typically used when:

- The system is in a multi-user state (for example, run-level 2).
- The system administrator is *not* the only person logged in and using the system.

In addition to shutting down HP-UX, `shutdown` can also bring the system to run-level s (single-user administration level).

For details on shutting down HP-UX using `shutdown`, see the *System Administration Tasks* manual and *shutdown*(1M) in the *HP-UX Reference*.

When to Use reboot

The `reboot` command shuts down all processes very quickly. It uses **`kill -9`** to kill any running processes. This can be dangerous (for example, cause loss of data) because `reboot` will shutdown processes immediately without letting them terminate normally. It is typically used when:

3

- The system is in run-level s (single-user administration level). (For a description of run-level s and how to bring the system to run-level s, see Chapter 6 and the *System Administration Tasks* manual.)
- You need to bring the system down very quickly.

Like **`shutdown`**, `reboot` allows you to specify a grace period. But unlike **`shutdown`**, when the grace period is over, `reboot` still kills processes quickly and indiscriminately (with **`kill -9`**).

For details on shutting down HP-UX using `reboot`, see the *System Administration Tasks* manual. For details on what `reboot` does, see *reboot*(1M) in the *HP-UX Reference*.

Examples

Table 3-1. Examples of using `shutdown` and `reboot`

Example	Description
`shutdown -h`	Shutdown and halt the system; give users a (default) 60-second grace period in which to logoff.
`shutdown -h 360`	Shutdown and halt the system; give users a 6-minute (360-second) grace period in which to logoff.
`shutdown -r`	Shutdown and reboot the system; give users the default 1-minute (60-second) grace period in which to logoff.
`reboot -h`	Shutdown and halt the system immediately.
`reboot`	Shutdown and reboot immediately.
`reboot -t +5`	Shutdown and reboot in five (5) minutes.
`reboot -t 22:05 -m "please logoff"`	Shutdown and reboot at 10:05PM. As 10:05PM approaches, display the message "please logoff" at more and more frequent intervals.

4

Login

Login is the means by which users gain access to HP-UX. *Beginner's Guide to HP-UX* and *Using HP-UX* describe *how* to login. This chapter describes *what* HP-UX does during login—that is, what happens between the time you type your username and the time you get a shell prompt. Understanding this procedure, you can modify it for your unique needs. For example, you can:

- Set up logins so that a user gets an application instead of a shell.
- Change all users' default login environments.
- Change all users' default login shells.

Such tasks can be done manually (for example, by manipulating system files and running various commands) or through the SAM program, which automatically updates system files and calls appropriate commands for you. For details on doing various tasks related to login, see the *System Administration Tasks* manual.

Note This chapter does not explore the intricacies of logging into a workstation implementing HP Visual User Environment (VUE). VUE has its own display and other environmental requirements, and thus reads its own configuration files. For information on VUE, consult the *HP Visual User Environment (VUE) System Administration Manual*.

Overview of Login

Login begins with the **/etc/getty** process spawned by the **init** program. The **getty** program prompts the user for a username (**login:**) and password, information which corresponds to the user's entry in the file **/etc/passwd**. **getty** passes the username and password to **login**, which verifies the user's validity on the system. **login** then allows access to the valid user and starts a shell.

Figure 4-1 shows a sample entry in **/etc/passwd** for a user named **terry** whose user ID is 265, group ID is 20, home directory is **/users/terry**, and login shell is **/bin/ksh**. You might wish to refer to this figure when reading the following section. For details on the **/etc/passwd** file, see "The /etc/passwd File" later in this chapter.

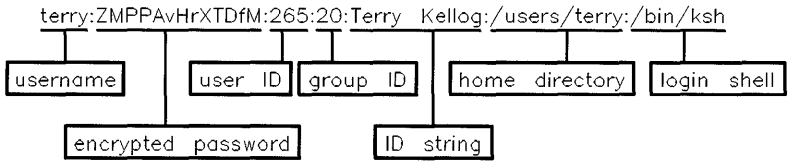


Figure 4-1. Sample /etc/passwd Entry for User terry

The login Process

The `login` process, described next, is illustrated in Figure 4-2.

1. `login` searches the `/etc/passwd` file for the username.
   a. If the username exists, `login` goes to step 2.
   b. If the username does *not* exist, `login`:
      i. Prompts the user for a password (`Password:`).
      ii. Displays the message `Login incorrect`.
      iii. Updates the `/etc/btmp` file (if it exists), which tracks invalid login attempts.
      iv. If the user attempts three consecutive invalid logins, `login` exits; otherwise, `login` prompts the user (`login:`) for a username and repeats step 1.
2. `login` checks to see if the username's password field is set in `/etc/passwd`.
   a. If so, it prompts the user for a password (`Password:`) and goes to step 3.
   b. If not, the user need not enter a password; go to step 4.
3. `login` compares the password to the username's encrypted password in `/etc/passwd`.
   a. If the password matches, `login` goes to step 4.
   b. If the password does *not* match, `login` displays the message "`Login incorrect.`" If the user attempts three consecutive invalid logins, `login` terminates; otherwise, `login` prompts the user for a username (`login:`) and control passes back to step 1.
4. `login` sets the username's user ID, group ID, and home directory from the corresponding fields in `/etc/passwd` (see Figure 4-1). `login` also updates the `/etc/wtmp` file, which tracks valid logins.
5. `login` executes the command specified in the command field of `/etc/passwd` (see Figure 4-1). Typically, the path name of the user's preferred shell. If the command field is empty, `login` starts a Bourne shell by default.

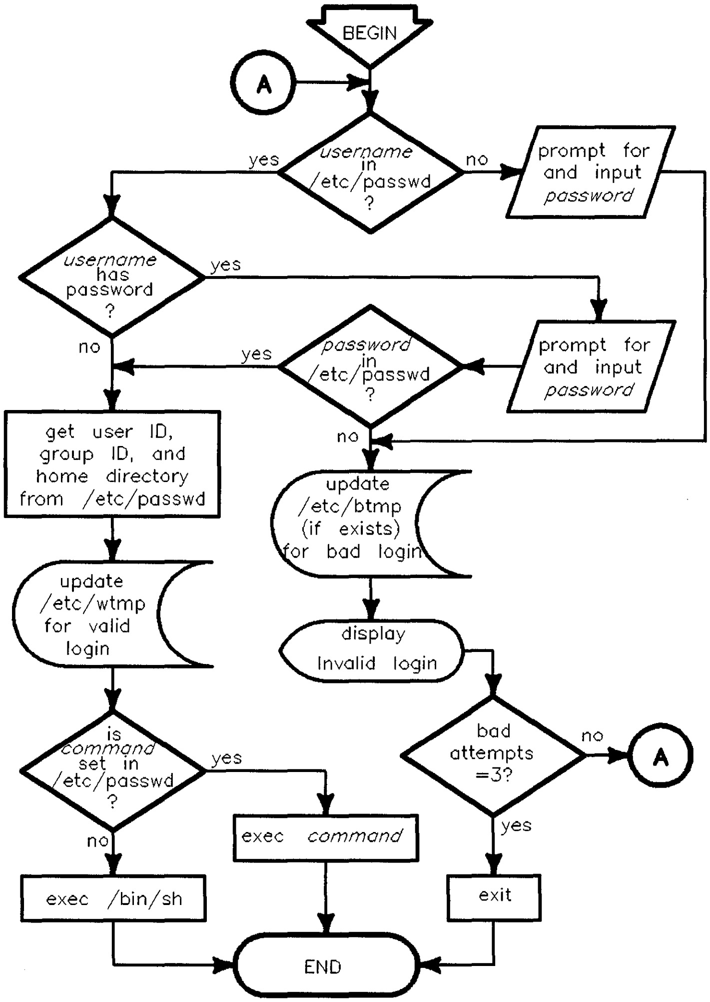


Figure 4-2. The `login` Process Control Flow

The Shell Environment Initialization Sequence

A shell initializes its environment before becoming available to a user:

1. The shell runs the appropriate **system login script**, which initializes the user's environment:

If the Shell Is ...	The System Login Script Is ...
Korn (`/bin/ksh`)	`/etc/profile`
Bourne (`/bin/sh`)	`/etc/profile`
Restricted (`/bin/rsh`, `/bin/krsh`)	`/etc/profile`
C (`/bin/csh`)	`/etc/csh.login`
Key (`/usr/bin/keysh`)	`/etc/profile`
PAM (`/bin/pam`)	none

   As shipped, these scripts define and export for shell use the environment variables `PATH`, `TZ`, and `TERM`. Since these scripts are run for all users at login, the system administrator can modify these files to set global defaults for all users.

2. The shell runs the user's **local login script** if it exists in the user's home (login) directory:

If the Shell Is ...	The Local Login Script Is ...
C (`/bin/csh`)	`.login`
Korn (`/bin/ksh`)	`.profile`
Bourne (`/bin/sh`)	`.profile`
Restricted (`/bin/rsh`, `/bin/krsh`)	`.profile`
Key (`/usr/bin/keysh`)	`.profile`
PAM (`/bin/pam`)	`.environ`

   Typically, the system administrator initially creates a local login script for each user. If used to add a user, SAM automatically creates a default local login script. Users can customize their environment by modifying these files to suit their needs. The Korn and C shells may have additional local login scripts:

a. *Korn shell*—If the `ENV` environment variable is defined, the Korn shell runs the file defined by `ENV` (typically, `.kshrc`) whenever a new Korn shell is started. Many programs (for example, `vi` and `mailx`) allow users to start a shell from within the program; this is called a **shell escape**. The `ENV` file is re-run for a shell escape, whereas `.profile` is run only at login.

b. *C shell*—Runs the `.cshrc` file whenever a new C shell is started. This is similar to how the Korn shell `ENV` file works.

c. *Key shell*—Runs the `.keyshrc` file whenever a new Key shell is started. If `.keyshrc` does not exist, `/usr/keysh/C/keyshrc` is used.

3. Once all initialization is complete, the shell displays a prompt and waits for input from the user.

Correcting Typing Mistakes during Login

During login, HP-UX does not know what kind of terminal the user is logging in on. Therefore, HP-UX has a primitive set of keys for editing. These key assignments are:

#	Does a backspace over the character just typed.

@	Kills the line (backspace to start of line).

Do not use these characters in usernames or passwords since they cannot be entered.

4

Once the user logs in, these key assignments are usually changed in the system or local login script via the **stty** command (see *stty*(1) in the *HP-UX Reference*), as follows:

```
stty erase ^H kill ^U
```

This sets the erase (erase previous character) key to Backspace and sets the kill (kill to start of line) key to CTRL U.

The /etc/passwd File

The `/etc/passwd` file contains essential information required during login. It contains entries (one per line) for all valid users of the system. Typically, the system administrator would modify or change these fields when adding new users to the system. As an easier alternative, the system administrator could use the SAM program to add users, which adds/changes the fields automatically (see the *System Administration Tasks* manual for details).

4

Syntax of Entries

The general form of `/etc/passwd` entries is:

username:*password*[,*pw_age*] :*userid*:*groupid*:*idstring*:*homedir*:*command*

: A colon. Fields are separated by colons.

username The username for which a user can login. `login` assigns this value to the environment variable `LOGNAME`. The `logname` command displays this value (see *logname*(1) in the *HP-UX Reference*).

password The encrypted form of the password the user must supply during login for this *username*.

If this field is empty (`::`), the user has no password and can login without typing one. This form of password is not recommended since it could lead to security problems.

Users can change their password without superuser privilege by using the `passwd` command (see *passwd*(1) in the *HP-UX Reference*).

Password aging, an additional security feature, can be added too, as described next under *,pw_age*.

Note On trusted systems, the password field contains only a `*`, and passwords are stored in a separate file, `/.secure/etc/passwd`. For details on trusted system, see *HP-UX System Security*.

,pw_age An optional aging field. Can be used to ensure that a user is forced to change his or her password every n-week interval, where n is

defined by the system administrator. This field is comprised of these parts:

,max min wks

There should be no spaces between the *max*, *min*, and *wks* parameters. They are shown separated here only to improve readability.

, The comma sets off the age fields from the encrypted password which precedes it.

max A single character representing the maximum number of weeks the user can use the password before being forced to use a new password. This character is encrypted as shown in Table 4-1:

4

Table 4-1. Password Aging Encryption Characters

Character	Weeks
.	0
/	1
`0` through `9`	2 through 11
`A` through `Z`	12 through 37
`a` through `z`	38 through 63

min A single character representing the minimum number of weeks the user must use the current password before being allowed to change it. Like the *max* character, the *min* character is encrypted, as shown in Table 4-1.

wks A string of encrypted characters defining the number of weeks (counted from the beginning of 1970) when the password was last changed. The `login` program uses and updates this field to enforce password aging. A null value is equivalent to specifying zero weeks.

If *max* and *min* are zero (e.g., *,pw_age* is `,..`), the user is forced to enter a password at the next login attempt and this field is removed. If *min* is greater than *max*, then only the system administrator can change the password.

See the password example for **terry** in the next section, "Sample /etc/passwd Entries".

userid A numeric user ID. Must be unique for each user. The **id** command displays this value (see *id*(1) in the *HP-UX Reference*).

groupid Group ID of the group to which the user belongs. Must be a valid group ID defined in **/etc/group**. The **id** command displays this value (see *id*(1) in the *HP-UX Reference*).

4

idstring A character string, typically holds the user's full name.

homedir Directory to use as the **home directory**—that is, the directory in which the user is initially placed after login. **login** assigns this value to the HOME environment variable.

command **login** runs *command* using the **exec** system call (see *exec*(2) in the *HP-UX Reference*). Any command can be placed in this field, but typically it contains the path name of a shell: **/bin/ksh**, **/bin/sh**, **/bin/csh**, or **/bin/pam**.

login assigns this value to the SHELL environment variable. Users can change their default shell without superuser privelege via the **chsh** (*ch*ange *sh*ell) command (see *chsh*(1) in the *HP-UX Reference*).

For more details on the format of the **/etc/passwd** file, (see *passwd*(4) in the *HP-UX Reference*).

Sample /etc/passwd Entries

`root:xE5/0qrnYf8Hg:0:1:System Administrator:/:/bin/sh`

Defines the root user (superuser) with an encrypted password, user ID 0, group ID 1, ID string "System Administrator", home directory **/** (root), and who uses the Bourne shell.

`michael:,..:125:10:Michael Moose:/users/michael:/bin/ksh`

Defines a user named `michael` who when he next tries to login, the system will force him to enter a new password (indicated by "`,..`" in the *password* field). In addition, he has user ID 125, group ID 10, id string "Michael Moose", home directory `/users/michael`, and he uses the Korn shell.

4

`terry:,9/:265:20:Terry Kellog:/users/terry:/bin/ksh`

Defines a user named `terry` who has password aging enabled. `login` will force `terry` to change her password every 11 weeks—indicated by the `9` as the *max* element of the *pw_age* field (`,`*9*`/`). In addition, `terry` cannot change her password more than once a week—indicated by the `/` as the *min* component of *pw_age* (`,9`*/*).

`guest:/bin/rsh`

Defines a `guest` user who has a restricted shell.

`who::90:1::/:/bin/who`

Defines a user named `who` with no password, user id 90, group id 1, home directory `/`, and command `/bin/who`. If a user attempts to login using username `who`, the system simply runs the `who` command and displays its output to the screen.

`date::91:1::/:/bin/date`

This is similar to the `who` entry. When the user types `date` to the `login:` prompt, the system runs the `date` command.

Login Tracking Files (/etc/btmp and /etc/wtmp)

Two files keep a log of logins: `/etc/btmp` keeps track of bad (failed) logins and `/etc/wtmp` keeps track of successful logins. In HP-UX diskless clusters, these files are context-dependent files (CDFs).

/etc/btmp

4

If `/etc/btmp` exists, the `login` process updates it automatically whenever a bad login attempt occurs. To create this file, issue the command:

```
touch /etc/btmp
```

Note, you must be `root` to write to the `/etc` directory.

Read the `/etc/btmp` file using the `lastb` command, to determine whether unauthorized users are attempting to login (see *last*(1) in the *HP-UX Reference*).

Note For security purposes, be sure `/etc/btmp` is not readable by users.

/etc/wtmp

The `login` process automatically updates `/etc/wtmp` whenever a user successfully logs in. Read this file using the `last` command (see *last*(1) in the *HP-UX Reference*).

Environment Variables

Environment variables define various characteristics of the environment in which your shell runs. Environment variables consist of a name and a value (`NAME=value`). Table 4-2 lists the environment variables used by the Korn, Bourne, and C shells. The assigned values of environment variables are used by your shell and passed to each process created during a session. For example, if you type the `cd` command without any options, it returns you to the directory contained in the HOME environment variable.

To see a list of all environment variables and their values defined in your shell environment, use the **env** command (see *env*(1) in the *HP-UX Reference*).

4

Note VUE reads different customization files to set environment variables than the standard HP-UX login scripts. Refer to the *HP Visual User Environment (VUE) System Administration Manual* for information on setting environment variables in VUE.

Table 4-2. Shell Environment Variables

Variable	Description	Default Value
LOGNAME	Contains the user's username. Set by **login**.	from **/etc/passwd**
HOME	Defines the user's home directory. Set by **login**.	from **/etc/passwd**
PATH	Lists the directories the system searches to find executable commands. Set by **login**.	**:/bin:/usr/bin**
SHELL	Contains the default shell. Set by **login** from *command* field of **/etc/passwd**. If there is no *password* field, the default is used.	**/bin/sh**
TERM	Specifies the kind of terminal the user is logged in on. Set by the local login script.	**hp**
TZ	Provides the current time zone and difference from Greenwich Mean Time. Set to Mountain Standard Time by your shell's default login script (the system administrator should change the value if you are in another time zone). TZ must be set *identically* in five different places: **/etc/rc**, **/etc/csh.login**, **/etc/profile**, **/etc/powerfail**, and (Series 300/400/700 only) **/etc/mkrs**.	MST7MDT
MAIL	Determines where the system looks for mail. Set by **login**, based on the username (for example, **/usr/mail/terry**).	**/usr/mail/$LOGNAME**

For details on setting and using environment variables, see *Using HP-UX*. The remainder of this section discusses two important environment variables: PATH and TERM.

The Command Search Path (PATH)

When you type a command, HP-UX searches all the directories specified by the PATH variable until it finds the command. If the command is not in a directory specified in PATH, the system displays:

command_name: `Command not found.`

If you or users of the system use commands in a particular directory frequently, you may want to change PATH to include this directory.

4

PATH Variable Format

The `PATH` variable contains a list of directories to search, separated by colons. There should be no spaces surrounding the colons. For example, suppose you use the `echo` command to determine the value of `PATH`, as follows:

```
$ echo $PATH
/bin:/usr/bin
```

This means that when you type a command, the shell first searches for the command in the `/bin` directory, then in the `/usr/bin` directory.

Commonly Used PATH Directories

Table 4-3 shows the path names of the most frequently used directories. You might want to add some (or all) of these directories to `PATH`.

Table 4-3. Possible Directories to Include in PATH

Directory	What It Contains
`/bin`	Frequently used HP-UX commands.
`/etc`	Commands the system administrator uses.
`/usr/bin`	Additional HP-UX commands.
`/usr/contrib/bin`	Contributed programs not supported by Hewlett-Packard.
`/usr/local/bin`	Programs and commands written locally (at your location).
`/usr/bin/X11`	X11 programs.
`$HOME/bin`	A directory you might create for your shell scripts and programs.
`.`	Commands and programs stored in your current directory. If `.` is in your `PATH`, it should be the last entry.

Remember that directories in PATH are searched in the order in which they appear (left to right). In general, put the most frequently used directories first in the path—unless two commands in the search path have the same name (for example, /bin/rm and $HOME/bin/rm). In this example, if you want the shell to find your version of rm first, put $HOME/bin/rm before /bin/rm in PATH.

Specification of Terminal Characteristics (TERM)

4

To communicate effectively with your terminal, HP-UX must know the type of terminal or graphics display you're using. The TERM environment variable, tset command, and stty command serve this purpose.

The default local login script prompts you to enter your terminal type as follows:

```
TERM = (hp)
```

Pressing (Return), sets the TERM environment variable to hp, the default value (this value works with Hewlett-Packard terminals, but it may not let you take full advantage of your terminal or graphics display features). Entering a different value, sets the TERM environment variable to that value.

Selecting a Value for the TERM Variable

HP-UX supports many terminal types . The /usr/lib/terminfo database tells HP-UX how to communicate with each terminal type. When you assign a value to TERM, the value must equal a value in the terminfo database (see *terminfo*(4) in the *HP-UX Reference*).

To illustrate, the files listed under usr/lib/terminfo/2 show all acceptable TERM values that begin with 2 (this is only a partial listing):

```
$ ls /usr/lib/terminfo/2
2382      2397a     2621a     2623p     2626-x40   2640a
2392      2500      2621k45   2624      2626A      2640b
2392A     2621      2621nl    2624a     2626P      2644
2392a     2621-48   2621nt    2624p     2626a      2645
2393      2621-ba   2621p     2625      2626p      2647
2393A     2621-fl   2621wl    2626      2627       2647F
⋮
```

Table 4-4 outlines the most common terminal and graphics display settings for Hewlett-Packard equipment. When more than one choice is listed, all choices are equivalent.

Note If you are using the HP Visual User Environment (VUE), HP recommends that you keep `TERM` set to `hp`. VUE has its own display requirements and thus looks in its own customization files for environment variables. Refer to the *HP Visual User Environment System Administration Manual* for information.

4

Table 4-4. Settings for the TERM Environment Variable

If You Are Using a ...	Set TERM to ...
terminal	the terminal's model number (for example, `2622`, `hp2622`, `262x`, `2392`)
Vectra	`2392`
medium resolution graphics display (512x600 pixels)	`3001` or `hp3001`
high resolution graphics display (1024x768 pixels)	`300h` or `hp300h`
HP 98550 display station (1280x1024 pixels)	`98550`, `hp98550`, `98550a`, or `hp98550a`
HP 98720 or HP 98721 display station (1280x1024 pixels)	The graphics interface card number (for example, `98720`, `hp98720`, `98731`, `hp98736a`)

Most users have a high-resolution terminal. The first setting you might try is `hp300h`. If that doesn't work, try the number of your graphics interface card (such as `987xx`). You can look in `/usr/lib/terminfo/9` for all the interface cards beginning with number `9`, or `/usr/lib/terminfo/h` for all interface cards beginning with `hp` to find your exact number.

Setting TERM with the tset Command

The tset command is a flexible command that sets the value of TERM and initializes your terminal characteristics. If you always log in using the same terminal type, you could change your local login script to eliminate the TERM prompt. In the local script, this command displays the TERM prompt:

```
eval ` tset -s -Q -m ':?hp' `
```

To customize the above command, replace ?hp with the value of TERM. For example, the following command initializes your terminal as a high-resolution graphics display (300h), but the TERM prompt itself does not display:

4

```
eval ` tset -s -Q -m ':300h' `
```

If you use more than one type of terminal (such as one at work and one at home), you could modify your tset command to include multiple terminal types. For an example, see "Sample C Shell .login Script".

Using stty to Modify and Display Terminal Characteristics

The stty command can be used to change various terminal characteristics, such as baud rate, erase and kill characters, etc. For example, the following stty command resets the erase and kill characters to # and @, respectively (the defaults during login):

```
$ stty erase # kill @
```

To change them back to the Backspace (CTRL H) and CTRL U keys, respectively, use:

```
$ stty erase ^H kill ^U
```

It also displays terminal settings if invoked with the -a option; for example:

```
$ stty -a
  speed 9600 baud; line = 0; susp = ^@; dsusp = ^@
  intr = ^C; quit = ^\; erase = ^H; kill = ^U; eof = ^D; swtch = ^@; eol = ^@
  -parenb -parodd cs8 -cstopb hupcl cread clocal -loblk -crts
  -ignbrk brkint ignpar -parmrk -inpck -istrip -inlcr -igncr icrnl -iuclc
  ixon ixany ixoff ienqak
  isig icanon -xcase echo echoe echok -echonl -noflsh
  opost -olcuc onlcr -ocrnl -onocr -onlret -ofill -ofdel -tostop
```

For details on the use and output of stty, see *stty*(1) in the *HP-UX Reference*.

Specifying the Working Environment with System Login Scripts

System login scripts contain commands to set a user's environment; for example, commands may check mail, set a user's terminal settings, and set environment variables. System login scripts run *before* local login scripts. This section describes the system login scripts `/etc/profile` and `/etc/csh.login`.

The system login scripts define a *default* environment. Local login scripts can override or modify these defaults for each user. Thus, the local login script provides a way for individual users to further modify their working environment.

4

Note The `/etc/profile` and `/etc/csh.login` scripts shown here may differ somewhat from those on your system. The main point to be gained from looking at these examples is to understand, generally, what these scripts do.

/etc/profile

`/etc/profile` is the default system login script for the Bourne, Korn, and Key shells. Its features are described after this example:

```
# @(#) $Revision: 66.6 $
# Default system-wide profile file (/bin/sh initialization).
# This should be kept to the bare minimum every user needs.
     trap "" 1 2 3                         # ignore HUP, INT, QUIT now.

     PATH=/bin:/usr/bin:/usr/contrib/bin:/usr/local/bin  # default path.
     MANPATH=/usr/man:/usr/contrib/man:/usr/local/man    # default path.
     if [ -r /etc/src.sh ]
     then
  . /etc/src.sh                     # set the timezone
        unset SYSTEM_NAME
     else
        TZ=MST7MDT                         # change this for local time.
        export TZ
     fi
     if [ "$TERM" = "" ]                   # if term is not set,
     then
             TERM=hp                       # default the terminal type
     fi
     export PATH MANPATH TERM
     stty ^H                               # Set erase to ^H
     trap "echo logout" 0
# This is to meet legal requirements...
     cat /etc/copyright
     if [ -r /etc/motd ]
     then
        cat /etc/motd                      # message of the day.
     fi
     if [ -f /bin/mail ]                   # notify if mail.
     then
             if mail -e
             then    echo "You have mail."
             fi
     fi
     if [ -f /usr/bin/news ]
     then news -n                          # notify if new news.
     fi
     if [ -r /tmp/changetape ]             # might wish to delete this:
     then    echo "\007\nYou are the first to log in since backup:"
             echo "Please change the backup tape.\n"
             rm -f /tmp/changetape
     fi
     trap 1 2 3                            # leave defaults in user environment.
```

Task	Explanation
Ignore signals	The opening **trap** instruction prevents the login script from being interrupted, by setting the following signals to be ignored: SIGHUP (hangup, 1), SIGINT (interrupt, 2), and SIGQUIT (quit, 3). Signals are documented in *signal*(5) of the *HP-UX Reference*.
Set PATH	Your shell searches the default PATH through **/bin**, **/usr/bin**, **/usr/contrib/bin**, and **/usr/local/bin** for the commands you issue. Your current directory is not included. Likewise, MANPATH identifies the locations of on-line specifications of the *HP-UX Reference Manual*.
Set time zone (TZ)	The TZ variable is set to Mountain Standard Time; change the setting in **/etc/profile** for other time zones.
Set TERM	If the TERM variable is not set, this script sets it to **hp**.
Export global variables	The **export** command passes the values of shell variables to subshells. This script exports PATH, MANPATH, TZ, and TERM.
Set **stty**	Sets terminal port to recognize ^H as the erase character.
Trap logout	This **trap** captures the exit signal (0) generated when you log out. When the shell receives this signal, the command "**echo logout**" is executed.
Display **copyright** file	The **cat** command displays the contents of **/etc/copyright**, a file of copyright messages required by law.
Display **motd** file	The **cat** command displays the contents of **/etc/motd**. This file contains the message of the day.
Check for mail	If the **/bin/mail** file exists, **mail -e** checks for mail. If you have mail, "**You have mail**" is displayed.
List news items	If the **/usr/bin/news** file exists, **news -n** is executed to list the news files in **/usr/news**.
Check for backup tape	If the file **/tmp/changetape** exists and is readable, the message, "**Please change the backup tape**" is displayed, instructing you to remove the tape from the tape drive.
Reset signals	The final **trap** resets signals 1, 2, and 3, after which, you can use hangup, interrupt, and quit.

/etc/csh.login

`/etc/csh.login` is maintained by the system administrator as the default system login script for the C shell. This section presents a sample script, and then explains its contents.

```
# @(#) $Revision: 66.4 $
# Default (example of) system-wide profile file (/bin/csh initialization).
# This should be kept to the bare minimum every user needs.

     # default path for all users.
     set path=(/bin /usr/bin /usr/contrib/bin /usr/local/bin)
     set prompt="[\!] % "
     # default MANPATH
     setenv MANPATH /usr/man:/usr/contrib/man:/usr/local/man

     if ( -r /etc/src.csh ) then
  source /etc/src.csh      # set the TZ variable
     else
  setenv TZ MST&MDT      # change this for local time.
     endif

     if ( ! $?TERM ) then                  # if TERM is not set,
             setenv TERM hp                #   use the default
     endif

# This is to meet legal requirements...
     cat /etc/copyright                    # copyright message.
# Miscellaneous shell-only actions:
     if ( -f /etc/motd ) then
             cat /etc/motd                 # message of the day.
     endif

     if ( -f /bin/mail ) then
             mail -e                       # notify if mail.
             if ( $status == 0 ) echo  "You have mail."
     endif

     if ( -f /usr/bin/news ) then
             news -n                       # notify if new news.
     endif

     if ( -r /tmp/changetape ) then        # might wish to delete this:
             echo
             echo "You are the first to log in since backup:"
             echo "Please change the backup tape.\n"

             rm -f /tmp/changetape
     endif
```

Task	Explanation
Set **path**	The **set** built-in command of the C shell sets **path** as a local shell variable. The C shell automatically sets the environment variable **PATH** to the shell variable **path**. The default **path** tells your shell to search **/bin**, **/usr/bin**, **/usr/contrib/bin**, and **/usr/local/bin** for source when you issue a command. The current directory is not included.
Set **prompt**	The shell variable **prompt** is a local shell variable in the C shell. This script sets **prompt** to "[*n*] % ", where *n* is the command event number. Command event numbers are used with the C shell's command history feature.
setenv MANPATH	The **setenv** built-in command of the C shell sets the environment variable **MANPATH** to search the paths listed for on-line specifications of the *HP-UX Reference Manual.*
Set time zone (TZ)	The **TZ** variable is set by default to Mountain Standard Time, but the contents of **/etc/src.csh** override it.
Set TERM	If the TERM variable is not set, this script sets it to **hp**.
Display **copyright** file	The **cat** command displays the contents of **/etc/copyright**, a file of copyright messages required by law.
Display **motd** file	The **cat** command displays the contents of **/etc/motd**. This file contains the message of the day.
Check for mail	If the file **/bin/mail** exists, **mail -e** checks for mail. If you have mail, "**You have mail**" is displayed.
List news items	If the **/usr/bin/news** file exists, **news -n** displays the list of news files in **/usr/news**.
Check for backup tape	If the file **/tmp/changetape** exists and is readable, the message, "**Please change the backup tape**" is displayed, instructing you to remove the tape from the tape drive.

4

Default Local Login Scripts

When adding a new user, the system administrator should make sure that the new user has a local login script. HP-UX provides three default local login scripts:

- /etc/d.profile for Korn and Bourne shell users.
- /etc/d.login and /etc/d.cshrc for C shell users.

4

When adding a new user manually, the administrator copies the appropriate file into the user's directory. For example, if the new user's shell is **ksh** and home directory is **/users/newuser**, the administrator copies as follows:

```
$ cp /etc/d.profile /users/newuser/.profile
```

If the administrator uses the SAM program to add a new user, SAM automatically copies the appropriate default local login script into the user's home directory.

/etc/d.profile

The default local login script resets the value of PATH, and sets terminal characteristics, the shell environment, and shell variables. This section shows a sample d.profile and describes its contents.

```
# @(#) $Revision: 66.1 $
# Default user .profile file (/bin/sh initialization).
# Set up the terminal:
    eval ` tset -s -Q -m ':?hp' `
    stty hupcl ixon ixoff
# Set up the search paths:
    PATH=$PATH:.
# Set up the shell environment:
    set -u
    trap "echo 'logout'" 0
# Set up the shell variables:
    EDITOR=vi
    export EDITOR
```

Task	Explanation
Set terminal characteristics	The `tset` command generates the `TERM = (hp)` prompt during login. Based on the value you enter, the TERM environment variable is set and the terminal's characteristics are defined. If you press Return, `TERM` is set to `hp`. Refer to *tset*(1) for more information. The `stty` command sets options for modem connections and scrolling text on your screen. Refer to *stty*(1) for more information.
Set PATH	Changes the value of `PATH` by adding the current directory to the end of `PATH`.
Set shell environment	The `set -u` command prevents the shell from attempting to execute command lines with undefined shell variables. This command is a safety feature because, by default, the shell substitutes a null value for variables that aren't defined. For example, if a script uses a variable named `WORK` and you have not assigned a value to `WORK`, the script will not execute. The `trap` command captures the exit signal (0) that is generated when you log out. When the shell receives this signal, the command "`echo logout`" is executed. (This `trap` command duplicates the one set in `/etc/profile`.)
Set shell variables	Shell variables such as `EDITOR` are set and exported as global variables. `EDITOR` is a Korn shell variable that determines which in-line editor the shell uses.

4

/etc/d.login

The default C shell local login script is similar to the Bourne shell/Korn shell script. The default local login script resets the value of PATH, sets terminal characteristics, and sets shell variables.

```
# @(#) $Revision: 64.2 $

# Default user .login file ( /bin/csh initialization )

# Set up the default search paths:
  set path=(/bin /usr/bin /usr/contrib/bin /usr/local/bin .)

# Set up the terminal type:
  eval `tset -s -Q -m ':?hp' `
  stty erase "^H" kill "^U" intr "^C" eof "^D"
  stty susp "^Z" hupcl ixon ixoff tostop
  tabs

# Set up shell environment:
  set noclobber
  set history=20
```

Task	Explanation
Set **path**	The value of **path**, if set here, explicitly changes the default search path. Note that for security purposes, the current directory should be placed at the end of the search path.
Set terminal characteristics	Because of the question mark in the string '`:?hp`', **tset** queries you for verification of **TERM = (hp)** during login. If you press (Return), **TERM** is set to **hp**. If you enter a different value, the **TERM** environment variable is set accordingly. See *tset*(1) for information on the **tset** command and *terminfo*(4) on the terminal capabilities database. The **stty** command sets options for modem connections and scrolling text on your screen. Refer to *stty*(1) for more information.
Set shell variables	The local shell variables **noclobber** and **history** are set. The shell variable **noclobber** restricts output redirection, to prevent accidental destruction of file contents. The shell variable **history** creates your command history buffer and sets its size. By default, **history** is set to record the last 20 commands executed. If this variable is not set, you have no command history and cannot edit and reuse previously executed commands.

The `/etc/d.login` (used by C shell) differs slightly in syntax and function from `/etc/d.profile` (used by Bourne and Korn shells). For information you can use to customize users' shells, consult *csh*(1), *ksh*(1), and *sh*(1) in the *HP-UX Reference Manual*

/etc/d.cshrc

The /etc/d.cshrc default script sets a restricted version of path for your subshells, sets local shell variables, and defines command aliases.

```
# Default user .cshrc file (/bin/csh initialization).

# Usage:  Copy this file to a user's home directory and edit it to
# customize it to taste.  It is run by csh each time it starts up.

# Set up default command search path:
#
# (For security, this default is a minimal set.)

     set path=( /bin /usr/bin )

# Set up C shell environment:

     if ( $?prompt ) then            # shell is interactive.
         set history=20              # previous commands to remember.
         set savehist=20             # number to save across sessions.
         set system=`hostname`       # name of this system.
         set prompt = "$system \!: " # command prompt.

         # Sample alias:

         alias       h       history
     endif
```

Task	Explanation
Set PATH	A restricted **path** is defined that includes only **/bin** and **/usr/bin**. Since **.cshrc** executes before **.login** during the login process, this restricted value of **path** is overridden by the path defined in **$HOME/.login**. When subshells are spawned, the restricted value of **path** is used.
Set shell variables	If the shell is interactive, various shell variables are set. The shell variable **history** creates your command history buffer and sets its size. In this script, **history** is set to record the last 20 commands executed. If this variable is not set, your subshells have no command history and you cannot edit and reuse previously executed commands. The shell variable **savehist** tells the shell to record the last 20 commands in the **$HOME/.history** file. This file is used as the history from which the next session starts. A shell variable **system** is set to the value output by the **hostname** command. The shell variable **prompt** is set to the value in the **system** variable, followed by the command event number and a colon.
Set command aliases	If the shell is interactive, command aliases are set. A command alias is an abbreviation for a long command line. This script sets an alias of **h** for the **history** command. The **/etc/d.cshrc** script also contains sample aliases that are commented out by default. Because of space limitations, these commands are not shown in the example. To use these aliases, remove the pound sign (#) at the beginning of the line, thus changing the line from a comment to a command.

Sample Bourne Shell .profile Script

The Bourne shell is the simplest shell in which to customize your environment. Only one login script is used, the `$HOME/.profile` script. This section first presents a sample `.profile` script, and then explains the script commands.

In `.profile`, exported variables are passed to all subshells. Variables that are not exported are used only by the login shell. The following script sets terminal characteristics, global environment variables, and local shell variables used only by the login shell. The script also sets up a logout script and displays some system status information.

```
# Set terminal characteristics:
eval ` tset -s -Q -m ':hp' `
tabs -T$TERM

# Set PATH:
PATH=$HOME/mybin:/bin:/usr/bin:/usr/contrib/bin:/usr/local/bin:

# Export global variables:
export PATH

# Set local shell variables:
PS1='$LOGNAME $ '
CDPATH=.:..:$HOME

# Set up logout script:
trap "echo logout; $HOME/.logout" 0

# Display system status information:
date
echo Currently `who | wc -l` users logged in.
```

Task	Explanation
Set terminal characteristics	This `tset` command sets and exports the `TERM` environment variable for a default HP terminal. The `tset` command also initializes the terminal's characteristics. The `tabs` command with the `-T` sets the tabs to the default format for your terminal.
Set PATH	This example sets PATH explicitly to search `$HOME/mybin`, `/bin`, `/usr/bin`, `/usr/contrib/bin`, `/usr/local/bin`, and the current directory. A colon (`:`) separates each directory. The trailing colon specifies the current directory.
Export global variables	The `export` command allows shell variables to be passed to all commands that you execute. In this script, PATH is exported.
Set local shell variables	The Bourne shell has several pre-defined shell variables. This script sets two: `PS1` and `CDPATH`. (These variables could also be exported to become global variables.) `PS1` is the primary shell prompt. Its default value is "`$ `". In this script, `PS1` is changed to your username, followed by a dollar sign and a space. `CDPATH` sets the search path for the `cd` command. A colon (`:`) separates each directory. As set here, the `cd` command searches for the named directory in the current directory (`.`), the parent directory (`..`), and the home directory (`$HOME`).
Set up logout script	The Bourne shell does not check for a `.logout` script automatically. The `trap` command captures the exit signal (0) and executes the specified commands. In this example, the `trap` echoes "`logout`" and executes a logout shell script when you log out.
Display system status information	The `date` command displays the current date and time. The `'who | wc -l'` pipe counts the number of users logged in and displays the results in the echoed string.

Sample C Shell .login Script

If the C shell is your login shell, HP-UX will look for a `.cshrc` script and a `.login` script in your home directory. The `.login` script is read after the `.cshrc` script when you log in. The `.login` script sets global environment variables. This section first presents a sample C shell `.login` script, and then explains the script commands.

Like the Korn shell, the C shell distinguishes between environment variables and shell variables. Environment variables are used by your login shell and are passed to subshells. Environment variables are defined in the `.login` script, and shell variables are defined in the `.cshrc` script.

The following script sets terminal characteristics, the global environment variable `PATH`, and the local shell variable `prompt`. The script also displays a login message.

```
% cat .login
# Set up the terminal type:
eval `tset -s -Q -m 'console:300h' -m '=1200:pc'`

# Set PATH:
set path=(/bin /usr/bin /usr/local/lib usr/lib . $HOME/mybin)

# Set local shell variables:
set prompt="[\!] "

# Provide login message:
echo `grep $LOGNAME /etc/passwd | cut -d: -f5` -- have a nice day.
```

Task	Explanation
Set terminal characteristics	The **tset** command sets up your terminal when you first log in to an HP-UX system. With the **tset** command, the login script can select between two or more terminal types. This example chooses between a high-resolution graphics display station (**300h**) that functions as the console or a personal computer (**pc**) logging in via a 1200 baud modem. To modify this command for your own use, replace **console** and **=1200** with the descriptions you need. For example, you might set one option to **=9600** (equals 9600 baud) and the other to **<9600** (less than 9600 baud). Then replace the **300h** and **pc** with the terminal types you use. (Refer to *tset*(1) for more information).
Set **PATH**	The **set** command sets local environment variables. **PATH** is a global variable, and **path** is a local variable in the C shell. The C shell automatically sets **PATH** equal to **path**. This example explicitly sets the entire value of **path**. The dot (.) represents the current directory.
Set local shell variables	The **set prompt** command redefines the default prompt (%) to show the event number being typed. This feature is useful with the command history mechanism. As set here the prompt will be "[*n*] " where *n* is the event number.
Provide login messages	The **echo** commands write messages onto the terminal screen. This message echos the user's name and the message "have a nice day." The user's name is extracted from the /etc/passwd file. The **grep** command finds the entry that contains **$LOGNAME**. The **cut** command extracts the fifth field which should be the user's name.

Key Shell—keysh

The Key shell (`keysh`) is a menu-based, context-sensitive softkey interface to the Korn Shell (`ksh`). With Key shell, you can perform most HP-UX user tasks with softkeys, rather than command-line entries. Key shell also provides online help for building softkey commands. Full information on using Key shell is found in the following locations:

- *keysh*(1) in the *HP-UX Reference Manual.*
- *Shells: User's Guide*

5

Process Management

Understanding how HP-UX manages processes will help you better interpret how your system carries out its computations. This chapter discusses:

- What a process is.
- How processes are created.
- How processes are killed.
- Commands for managing processes.
- How the kernel manages processes.
- HP-UX multiprocessing.

What Is a Process?

A **process** is a running program, managed by such system components as the scheduler and the memory management subsystem. Although processes appear to the user to run simultaneously, in fact a single processor is executing only one process at any given moment.

Described most simply, a process consists of text (the code that the process runs), data (used by the code), and stack (a "place" in the kernel where the parts of a program are stored as the process is running). Two stacks are associated with a process—kernel stack and user stack. The process uses the user stack when in user space and the kernel stack when in kernel space.

In addition, a process includes:

- The program's data structures (variables, arrays, records).
- A process ID, parent process ID, and process group ID.
- The process's user and group IDs (both real and effective IDs).
- A group access list.
- Information on the process's open files.
- The process's current working directory.
- An audit ID (on trusted systems only).

Process Relationships

Processes maintain hierarchical, parent-child relationships. Every process has one parent, but a parent process can have many child processes. Processes can create processes, which in turn, can create more processes. The relationships among processes can be represented by a tree graph. For example, Figure 5-1 shows the relationships among processes `P1`, `P2`, `P3`, `P4`, `P5`, and `P6`.

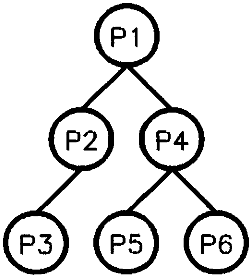


Figure 5-1. Process Tree Graph

In this graph, you can see the following relationships:

- P1 created P2 and P4. Therefore, P1 is the parent of P2 and P4; P2 and P4 are the children of P1.
- P2 created P3. Therefore, P2 is the parent of P3; P3 is the child of P2.
- P4 created P5 and P6. Therefore, P4 is the parent of P5 and P6; P5 and P6 are the children of P4.
- P1 is the *grand*parent of P3, P5, and P6; P3, P5, and P6 are the *grand*children of P1.
- P3, P5, and P6 have not yet created any child processes.

A child process inherits its parent's environment (including environment variables, current working directory, open files). Also, all processes except system processes (such as `init`, `pagedaemon`, and `swapper`) belong to **process groups**. Process groups are explained later in this chapter.

Process and Parent Process IDs

When a process is created, HP-UX assigns the process a unique integer number known as a **process ID** (**PID**). The HP-UX kernel identifies each process by its process ID when executing commands and system calls.

A process also has a **parent process ID** (**PPID**), which is the PID of the parent process.

Using the **ps** command, you can see the process and parent process IDs of processes currently running on your system (see *ps*(1) in the *HP-UX*

Reference). For example, if your username were `terry`, you might see something like this when executing `ps`:

```
$ ps -f
     UID   PID  PPID  C    STIME TTY       TIME COMMAND
   terry  3865  3699  2 13:35:43 ttyp3     0:00 ps -f
   terry  3699  3698  0 12:58:21 ttyp3     0:00 ksh
```

This indicates that the user `terry` has two processes running, the `ps -f` command and `ksh` (the Korn shell). Notice that the PPID of the `ps -f` command is the same as the PID of `ksh`. This is because `ps -f` was spawned from the `ksh` command line; thus, `ksh` is the parent of `ps -f`, and `ps -f` is the child of `ksh`. (For details on the other columns shown, see *ps*(1) in the *HP-UX Reference*.

5

User and Group IDs (Real and Effective)

In addition to the process ID, a process has other identification numbers:

- a real user ID
- a real group ID
- effective user ID
- effective group ID.

A **real user ID** is an integer value that identifies the owner of the process—that is, the user ID of the user who invoked the process. A **real group ID** is an integer value that identifies the group to which the user belongs. The real group ID is shared by all users who belong to the group. It allows members of the same group to share files, and to disallow access to users who do *not* belong to the group.

The *id*(1) command displays both integers and names associated with real user ID and real group ID. The `/etc/passwd` file assigns the real user ID and real group ID to the user at login. (For details, see Chapter 4, "Login".)

```
% id
uid=513(terry) gid=20(users)
% grep 513 /etc/passwd
terry:EqqHevH:513:20:Terry Ho,[44MY],495-0601,:/users/terry:/bin/csh
```

The **effective user ID** and **effective group ID** allow a process running a program to act as the program's owner while the program executes. The effective IDs are usually identical to the user's real IDs. However, the effective user ID and group ID can be set individually to protect a program, by making the effective IDs of the program's processes equal to the real IDs of the program's owner.

The effective IDs are used to allow a user to access or modify a data file or to execute a program in a limited manner. When the effective user ID is zero, the user is allowed to execute system calls as the superuser (described in the following section).

For example, suppose the dean of a university keeps all students' records in a file on the system. He wishes to allow an English professor to modify a student's record, but only for that professor's class. The dean first sets read and write permissions on the file containing the student's records. He then writes a program that takes as input the professor's modifications, verifies that the professor, as user, may change the record, then modifies the record if it is allowed. Finally, the dean protects the program by assigning the effective IDs of the user to the dean's real IDs when the program is executed. Thus, when the program accesses the student record file, the system allows the program to read from or write to the file because it believes that the dean is accessing the file (the effective user and group IDs are those of the dean). However, the real user and group IDs of the process remain the same as those of the user invoking the program. The program can use these IDs to verify access permission to the data (that is, ensuring that an English professor is accessing English class records).

The effective user and group IDs remain set until:

- The process terminates.
- The effective IDs are reset by an *overlaying* process, if the **setuid** or **setgid** bit is set. (See *exec*(2) in the *HP-UX Reference*.
- The effective, real, and saved IDs are reset by the system calls **setuid**, **setgid**, **setresuid**, or **setresgid**. (See section 2 in the *HP-UX Reference*).

Audit ID (Trusted Systems Only)

If you have converted to a trusted system, each user has an **audit ID**. This audit ID does not change, even when the user executes programs that use a different effective user ID. Refer to *HP-UX System Security* for more information.

Jobs

A **job** is any command or command pipeline invoked from an HP-UX shell. When you execute a command, the operating system interprets the entire task required by that command as a job.

Job Control

HP-UX supports **job control** for both the Korn shell and C shell. Job control provides users with greater flexibility in managing and controlling jobs. For example, you can:

- Temporarily stop (suspend) a foreground job, by pressing CTRL-Z. This can be customized using the `stty` command. see the *HP-UX Reference*.
- Bring a background job into the foreground, using the `fg` built-in shell command.
- Move a foreground job into the background, using the `bg` built-in shell command.

5

For more information, see *stty*(1), *csh*(1), *glossary*(9), and *ksh*(1) in the *HP-UX Reference*. Job control is also discussed in *Shells: User Guide* and *A Beginner's Guide to HP-UX*.

Process Groups

Every process (except system processes, such as `init` and `swapper`), belongs to a **process group**. When you create a job, the shell assigns all the processes in the job to the same process group. Signals can propagate to all processes in a process group; this is a principal advantage of job control. (Signals are discussed later in this chapter.)

Each process group is uniquely identified by an integer called a **process group ID**. Each process group also has a **process group leader**. The process group's ID is the same as the process ID of the process group leader. Every process in a process group has the same group ID.

A process group ID cannot be re-used by the system until its **process group lifetime** ends. The process group lifetime is a period of time beginning when a process group is created and ending when the last remaining process in the group leaves. A process leaves either:

- When another process executes a `wait()` or `waitpid()` function for an inactive process, or
- When calling the `setsid` or `setpgid` system calls (see *setsid*(2) and *setpgid*(2) in the *HP-UX Reference*).

Group Access Lists

Each process has a **group access list** of up to `NGROUPS_MAX` groups to which the process belongs. `NGROUPS_MAX` is defined in `/usr/include/limits.h`, and is typically 20. Programmatically, you can use `sysconf()` to obtain the value.

A process is permitted to access the files of any group in this list as though that group were the process's effective group ID. The access list is assigned at login based on the group memberships specified in the file `/etc/group` (Series 300/400/700) or `/etc/logingroup` (Series 800).

You can control group access by using the `chgrp` command. (See *chown*(1) in *HP-UX Reference Manual.*)

5

Sessions

Each process group is a member of a **session**. All processes started during a single login session belong to a session. Or, think of a session as a login shell with all the jobs the login shell spawns. On a system running windows or shell layers, each window or layer is a session.

A process belongs to the same session as its creator (parent). A process can alter its session affiliation using the `setsid` system call. The shell uses the `setpgid` system call to create jobs and ensure that all processes in a job belong to the same process group. (See *setsid*(2) and *setpgid*(2) in the *HP-UX Reference.*)

A **session leader** is the process that created the session via the `setsid` system call (normally the login shell). **Session lifetime** is the period between when a session is created and the end of the lifetime of all process groups that belong to the session.

A special instance of process groups is the **orphaned process group**. This is a process group in which the parent process of every member is either itself a member of the group or is not a member of the group's session. Terminal signals (such as ^Z) cannot stop an orphaned process group.

Processes and Terminal Affiliation

Every session has one **controlling terminal**. The session leader connected to the controlling terminal is called the **controlling process**. The exceptions to this are daemon processes (such as `cron` and `inetd`); they have no controlling terminal. All processes belonging to the session use the controlling terminal for standard input, standard output, and standard error.

At any one time, one process group in a session is the **foreground process group**. Typically, the foreground process group is the job that the login shell runs in the foreground, or the shell itself if no foreground job has been spawned. The foreground process group has primary access to the controlling terminal, beyond that of all **background process groups**. The foreground process group can read from and write to the controlling terminal without restrictions.

Attempted read by Background Process Group

If a process in a background process group attempts to read from its controlling terminal, its process group is signaled with SIGTTIN, which suspends the process to which it is sent, except in two instances:

- When the reading process ignores or blocks SIGTTIN
- When the reading process belongs to an orphaned process group.

In either case, the `read` returns −1 and no signal is sent.

Attempted Write by Background Process Group

If a process in a background process group attempts to write to the controlling terminal, its process group is signaled with SIGTTOU which, by default, stops (suspends) the process to which it is sent. There are three cases where SIGTTOU is not sent:

- If the TOSTOP terminal characteristic is *not* set, the process is allowed to write.

- If the `TOSTOP` terminal characteristic is *set*, and if the writing process ignores or blocks `SIGTTOU`, the process is allowed to write.
- If the `TOSTOP` characteristic is *set*, and the writing process belongs to an orphaned process group, and the writing process does not ignore or block `SIGTTOU`, the call to `write` returns −1.

Process Creation

A process can create another process to:

- concurrently execute another program.
- execute another program and wait for its completion.

At a shell prompt, you create a new process by typing the command's name and pressing (RETURN). The shell is the parent process; the process you create is the child process. The child process can then create its own child processes, of which it is the parent. 5

At the system programming level, a new process is created when a program calls either the `fork` or `vfork` system call. The parent process is the calling process; the child process is the created process. The `system` subroutine can also spawn a child process; `system`'s calling process waits for the child process to terminate before continuing.

The following section discusses briefly key system calls initiated when you run a program. For detailed information on the intricacies of processes and programs, consult the manual *Programming on HP-UX*, and *system*(3S), *fork*(2), *vfork*(2), and *exec*(2) in the *HP-UX Reference*.

The fork System Call

The `fork` system call creates ("forks") a new process.

In earlier implementations of HP-UX, the system copied the entire data segment of a process every time `fork` was issued. `fork` time increased as the size of the data and stack segments increased. Because processes can be larger than available memory, copy time could increase dramatically, degrading system performance.

HP-UX now implements two architecture-specific means by which the system can create processes more quickly—"copy-on-write" (Series 300/400) and "copy-on-access" (Series 700/800). These are described in Chapter 7, "Memory Management" of this manual.

When the parent process is written to the child address space, it differs from the parent process in its process ID and parent process ID); it has exact copies of the parent's code, data and current variable values. (See *fork*(2) in the *HP-UX Reference* for a detailed list of similarities and differences.)

When the `fork` system call is executed, the system must have enough free swap space to duplicate the parent process or `fork` fails. Once the child process is created, both processes begin to execute from the program statement immediately following the call to `fork`.

5

The `fork` system call returns the child's process ID (a non-zero value) to the parent process, while the identical call in the child's copy of the code always returns zero. Since the process IDs returned by `fork` are distinguishable, each process can determine whether it is the parent process or the child process.

For example, suppose a process consists of a program that tests the life of car batteries. The program has read 1000 data values from a voltmeter and is ready to print and plot the data. The program could have been written to do one task completely (such as printing the data) and then perform the second task (plotting the data). However, the programmer has included a `fork` system call in his program at a location after the data has been read.

When the program completes the statement containing the `fork` system call, two nearly identical processes exist. Each process examines the value returned by its `fork` system call to determine whether it is the child process or the parent process. Following the `fork` statement is a conditional branch statement that states, "If the process is the child process, print the data. If the process is the parent process, plot the data." The `fork` statement and the conditional branch statement enable both printing and plotting to be done simultaneously. And since each process has its own copy of the test data, each can modify the data without affecting the other process.

The vfork System Call

The `vfork` system call is functionally identical to `fork`, and provided only for backward compatibility.

The exec System Call

Often following the **fork** system call is an **exec** to another program. **exec** is a system call that overlays separate code and data on top of already existing process code and data. In this manner a parent process is able to create a new process using **fork**, and subsequently execute an entirely different program via exec.

For example, suppose you are editing a file and need to verify the name of another file. You might use a shell escape to **fork** a shell and **exec** the program ls.

Open Files

For a process to access files, it must first open them. A process inherits all open files from the parent. Three files that are usually open are **standard input** (stdin), **standard output** (stdout), and **standard error** (stderr). When a process terminates, the system closes any files opened by the process.

Two configurable operating-system parameters govern the number of open files permitted per process. The parameter **maxfiles** defines the soft limit, representing the maximum number of open files a process can have without issuing the *setrlimit*(2) system call. By default, **maxfiles** is set to 60.

The parameter **maxfiles_lim** defines the hard limit of open files permitted per process. By default, **maxfiles_lim** is set to 1024. Only a superuser process can increase this parameter beyond its hard limit. **maxfiles_lim** is used with *setrlimit*(2) to change number of open files permitted. **maxfiles_lim** can be increased up to a maximum of 2048 or decreased to a minimum of 60. (Programmatically, you can use **sysconf()** to get the current value.)

A process's soft limit can be increased up to the hard limit or decreased to any value greater than number of the highest file-descriptor allocated. An absolute limit of 2048 is imposed on both hard and soft limits. If the value of maxfiles_lim is 60, no one can increase their open-file (maxfiles) limit.

When considering the appropriate setting of soft and hard limits, consider too the operating-system parameters, **nfile** and **ninode**, because they are affected as well. (See Appendix A, "System Parameters" in the *System Administration Tasks* manual for descriptions of **nfile** and **ninode**.)

Process Termination

At the system programming level, a process **terminates** (dies) when:

- The process successfully finishes running.
- The process intentionally terminates itself by calling the `exit` system call (see *exit*(2) in the *HP-UX Reference*).
- The process receives a signal whose default action is fatal.

When a process dies, all its open files are closed and most of the resources the process holds are deallocated. The process then enters an inactive state in which it still holds some system resources. The remaining resources are returned to the system, the last being process's entry in the `proc` table. The process dies.

5

At the shell level, processes can be terminated via the `kill` command, described in more detail in the next section.

Process Management Commands

From both command level (from a shell) or from SAM (Process Management area), you can manage and monitor processes. The most useful tools are described in this section. HP-UX can also provide system accounting information for terminated processes (see Chapter 14, "System Accounting").

Understanding Process Status—ps

The `ps` command, run from a shell or from SAM, displays the following information about processes currently running on the system:

- User ID of the user who spawned the process
- Process ID
- Parent process ID
- Command line used to spawn the process
- `tty` (terminal) line from which command was invoked
- Length of time the process has been running (real CPU time).

`ps` also allows you to query the system about processes selectively. It is a very versatile command which is quite useful to system administrators and general users alike.

When invoked without options, `ps` displays the process ID, terminal ID (`tty`), real CPU time usage, and name of all commands a user is running. If the `-f` (*f*ull) option is specified, `ps` also displays login name, parent process ID, and time the process was `fork`ed.

```
$ ps -f
   UID    PID  PPID  C    STIME TTY      TIME COMMAND
mickey   3286  2016  9 16:19:03 ttyp1    0:00 ps -f
mickey  25705 25649  0 08:47:58 ttyp1    0:02 -ksh /users/michael [ksh]
mickey   2016 25705  0 15:13:02 ttyp1    0:24 vi processes.tag
```

Including the `-e` option causes `ps` to display information for *all* processes in the system, not just those of the user who invoked it:

```
$ ps -ef
   UID    PID  PPID  C    STIME TTY      TIME COMMAND
mickey  25737 25715 14 08:48:13 ttyp3    0:00 xcall /usr/local/bin/xcall
  root  13322     1  0  Jun  6  ?        0:00 cron
mickey  24357     1  0 19:45:46 console  0:01 -ksh [ksh]
  root      4     0  0  Jun  6  ?        7:15 netisr
     :
```

5

If the `-l` (*long*) option is specified, `ps` displays user ID, state (`S`), nice value (`NI`), memory or disk address of the process (`ADDR`), priority (`PRI`), and size (`SZ`) in blocks of the process core image. (State and priority are discussed in some detail later in this chapter.)

```
$ ps -l
F S   UID   PID  PPID  C PRI NI    ADDR   SZ    WCHAN TTY   TIME COMD
1 R   513 11009  7793  5 179 20  d6e200   16          ttyu4 0:00 ps
1 S     0  7792   133 15 154 20  e06100   13   214fb0 ttyu4 0:00 rlogind
1 S   513  7793  7792 16 168 20  df5a80   52 7ffe6000 ttyu4 0:00 csh
```

The `-l` option is useful for isolating problems on a sluggish system. For example, a runaway process might show up with an excessively large entry in the `TIME` column. You can trace the process back to its owner via the `UID` column. Similarly, if a process is not responding or getting time to run, the `S` (status) column might indicate whether the process is sleeping.

5 For details on using `ps`, see *ps*(1) in the *HP-UX Reference*.

Understanding Process Termination—kill

The `kill` command, run from a shell or from SAM, terminates a running process when you specify the process's PID. By default, `kill` terminates the process gracefully via the `SIGTERM` termination signal. "Graceful" means that the process first handles any additional signals pending. By specifying a signal number, you can kill processes more abruptly if necessary (for example, `kill -9`, where `9` specifies the `SIGKILL` signal). See *signal*(5) in the *HP-UX Reference* for complete description of the signals handled by the multiple signal interfaces supported by HP-UX.

For example, suppose a user is running a program that has hung the terminal; that is, the program does not respond to keyboard input from the user and the user cannot exit the program. As system administrator, you can kill the user's program from another terminal by doing the following:

1. Determine the process ID of the program, by running `ps`.
2. Kill the process via `kill`.

If the program was named `list_data` and the user's user ID was 265, you might see the following:

```
$ ps -f -u 265
     UID   PID  PPID  C    STIME TTY      TIME COMMAND
```

```
terry  3677  3638  0 12:27:29 ttyp1    0:53 list_data
terry  3638  3632  0 12:25:02 ttyp1    0:00 ksh
```

To kill the process gracefully, type:

```
$ kill 3677
```

This would, in most cases, kill the process and the user would again be able to use the terminal. If this does not work, you might try:

```
$ kill -9 3677
```

Note This form of `kill` should be used cautiously because it bears a higher risk of data loss.

Understanding Relative Process Priority—nice and renice

5

All processes have a priority, set when the process is invoked and based on factors such as who is running the process (user, system) and whether the process is created in a time-share or real-time environment.

The `nice` command can be used to set a process to run at a lower priority than would be set by default. `nice` does not lower the priority of an already running process. `nice` is useful for running programs whose execution time is not critical.

For example, suppose you have a program, named `numcrunch`, that manipulates large arrays of data, but the data is not critical to your work at the moment. How long it takes for the program to manipulate the data is unimportant; more critical programs should have greater access to CPU resources. To run `numcrunch` as a low priority background process, type:

```
$ nice numcrunch &
```

Note that both Korn and C shells handle `nice` slightly differently: `ksh` automatically lowers priority of background processes by four; this behavior can be modified using the `bgnice` argument. If you specify `nice` from `ksh`, it executes `/usr/bin/nice` and lowers priority by ten. If you specify `nice` from `csh`, it executes its built-in command and lowers priority by four; however, if you specify `/usr/bin/nice`, `csh` lowers priority by ten.

The `renice` command allows you to alter the priority of *running* processes. Running processes can also be altered from the Process Management area of SAM.

For details, see *nice*(1), *ksh*(1), *csh*(1), *renice*(1), and *nice*(2) in the *HP-UX Reference*.

Tools for Monitoring Process Management Performance

The following tools may provide additional information to help you examine your system's operation. Although they are commands, some (`top`, `monitor`, and `sar`) are accessible from the Process Management area of SAM:

top(1)	Displays and updates information about the top processes on the system. Summarizes the general state of the system (load average), quantifies amount of memory in use and free, and reports on individual processes active on the system. Whereas `ps` gives a single "snapshot" of the system, `top` samples the system and updates its display at intervals. On multi-processing systems, `top` reports on the state of each each CPU.
monitor(1M)	Displays a wide variety of system performance statistics, in each major system area (processes, memory, I/O, and networking).
sar(1)	(Series 800 only.) Reports on cumulative system activity, including CPU utilization, buffer activity, transfer of data to and from devices, terminal activity, number of specific system calls used, amount of swapping and switching activity, queue lengths, and other kernel tables.
vmstat(1)	Quantifies the use of virtual memory by processes on the system; also reports on traps and CPU activity.
iostat(1)	Reports I/O statistics for active disks, terminal, and processor.

Process Management and the Kernel

You can think of the process control system as containing the kernel's scheduling subsystem, memory management subsystem, and interprocess communication (IPC) subsystm.

The process control system interacts with the file system when reading files into memory before executing them. Several processes may be instances of the same program; for example, more than one person might be using `vi` at same time; each invocation of `vi` is an instance of the same process.

A process reads and writes its own data and stack, but cannot write/read any other processes'. (Note, however, shared memory can be read and written by several processes.)

Processes communicate with other processes via shared memory or system calls. Communication between processes (IPC) includes asynchronous signaling of events and synchronous transmission of messages between processes. System calls are requests by a process for some service from the kernel, such as I/O, process coordination, system status, and data exchange. All HP-UX system calls are documented in section 2 of *HP-UX Reference Manual*.

Much valuable information about CPU usage can be obtained by running `monitor` or `top` from the Process Management area of SAM.

Process Modes

An HP-UX process can execute in two modes—kernel or user mode—and through its process lifetime, switches between them. Information about the process (such as variables, process addresses, buffer counts) accumulates in a "stack" for each mode, and it is through these stacks that the process executes instructions and switches modes.

For Series 300/400, certain kinds of instructions trigger mode changes; for example, when a program invokes a system call, an entry point in the system call library is encoded in assembly language and contains "trap" instructions, which, when executed, cause an interrupt that results in a switch to kernel mode. When the process executes the instruction, it switches mode to the kernel, executes kernel code, and uses the kernel stack.

For Series 700/800, when you make a system call, you go through system call stub code, which passes the system call number through a gateway page that adjusts privilege bits to put the process in kernel mode.

Every process has an entry in a kernel process table and a `u_area` structure, which contains private data such as control and status information. The context of a process is defined by all the unique elements identifying it—the contents of its user and kernel stacks, values of its registers, data structures, and variables.

Although HP-UX gives the illusion of executing multiple processes simultaneously, each CPU actually executes one process at a time, according to its priority. (In multiprocessing systems, several SPUs can be executing a process simultaneously.)

5 The principle behind the distribution of CPU time is called **time-slice**—the amount of time a process can run before the kernel checks to see if there is an equal-priority process ready to run.

Process Priorities

Processes are chosen by the scheduler to execute a time-slice based on their **priority**. Priorities range from 0 (highest priority) to 256 (lowest priority), and are classified by need. You can see at what priority a process is set to run at by looking in the `PRI` column when you invoke `ps` or `top`. The following lists the three categories of priority:

Real-time priority (0-127)	Reserved for processes started with `rtprio()` system calls.
System priority (128-177)	Used by system processes.
User priority (178-256)	Assigned to user processes.

The kernel can alter the priority of time-share priorities (128-256) but not real-time priorities.

Run Queues

A process must be on a queue of runnable processes before the scheduler can choose it to run. Figure 5-2 shows an abstraction of run queue organization. Run queues are link-listed in decreasing priority. Each process is represented by its header on the list of run queue headers; each entry in the list of run queue headers points to the process table entry for its respective process. Processes get linked into the run queue based on the process's priority, set in the process table.

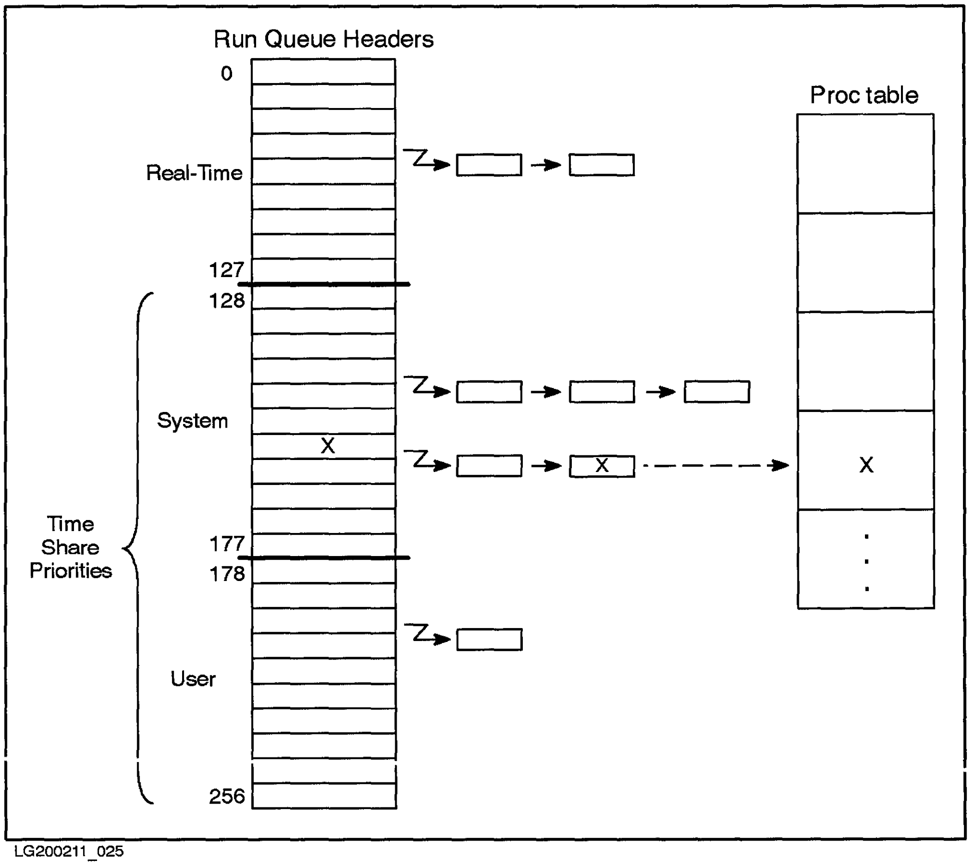


Figure 5-2. Run Queue, showing Priorities

5

The kernel maintains separate queues for system-mode and user-mode execution. When a timeshare process is not running, the kernel improves the process's priority (lowers its number). When a process is running, its priority worsens. The kernel does not alter priorities on real-time processes. Timeshared processes (both system and user) lose priority as they execute and regain priority when they do not execute.

The scheduler chooses the process with the highest priority to run for a given time-slice. system-mode priorities take precedence for CPU time. User-mode priorities can be preempted—stopped and swapped out to secondary storage; kernel-mode priorities cannot. Processes run until they have to wait for a resource (such as data from a disk, for example), until the kernel preempts them when their run time exceeds a time-slice limit, until an interrupt occurs, or until they exit. The scheduler then chooses a new eligible highest-priority process to run; eventually, the original process will run again when it has the highest priority of any runnable process.

Process Context

When the kernel switches from executing one process to executing another process, it switches **context**. That is, the kernel switches from executing in the context of one process to the context of another process. When this happens, the kernel saves information about the interrupted process to be able to later resume execution. The kernel also restores the context of the process it is about to start.

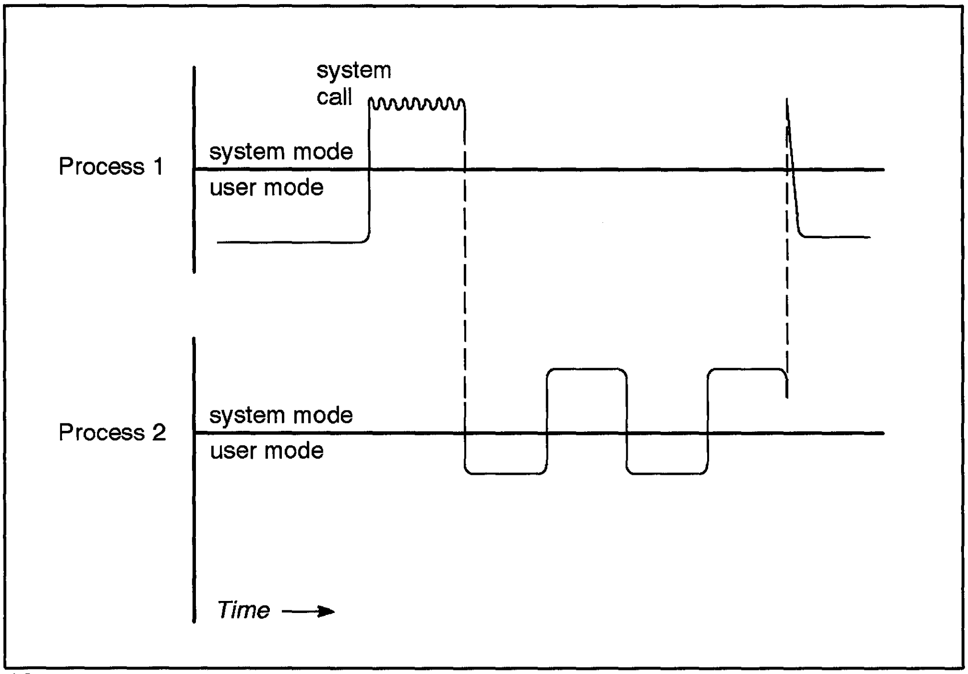


Figure 5-3. System Switching Context from One Process to Another

Context switching is shown in Figure 5-3. In this figure, the system switches from executing in the context of process P1 to executing in the context of process P2. The dotted vertical lines indicate the points of context switching. Note that context switching can only take place when the process is executing in system mode.

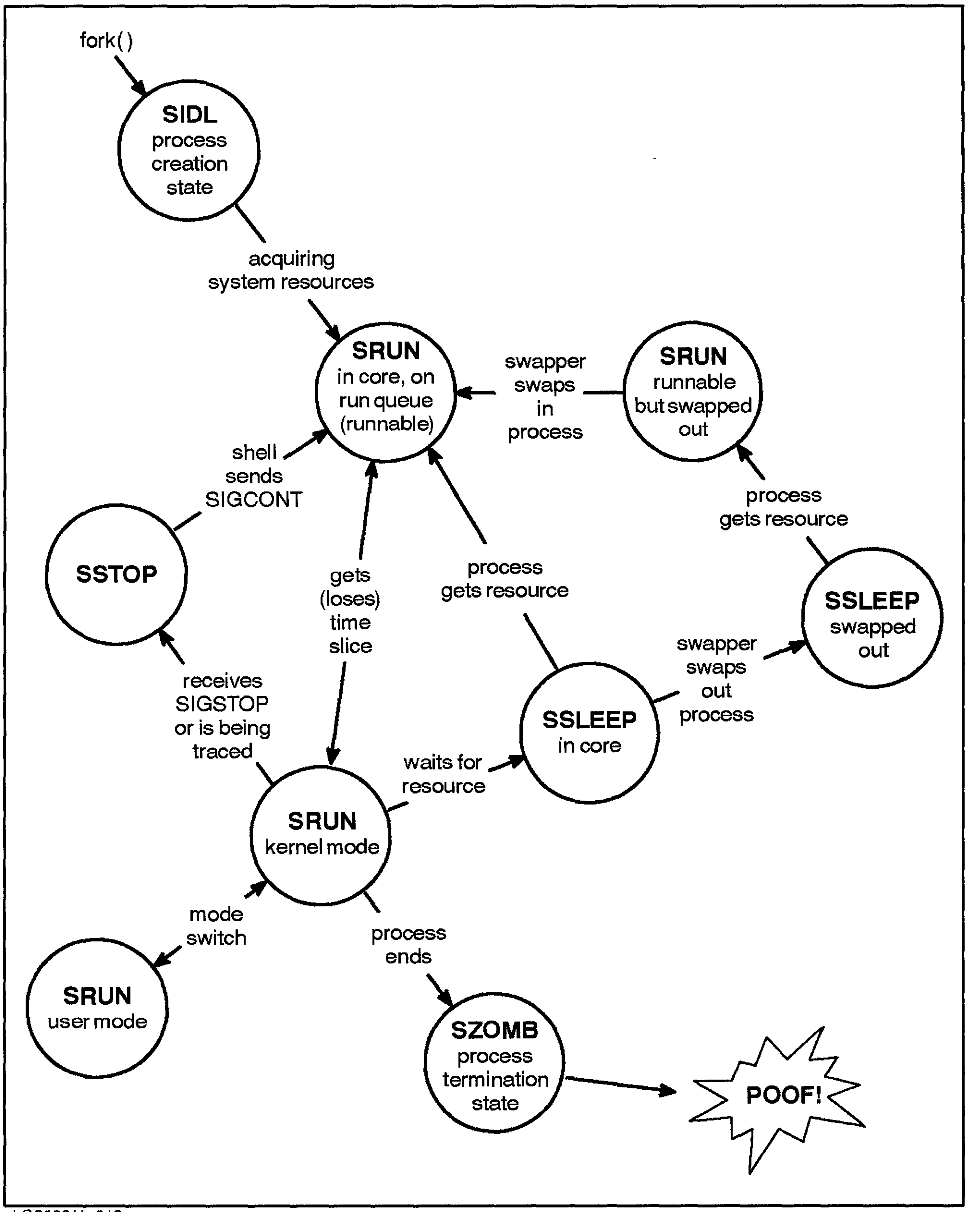
Process States and Transitions

Throughout the course of its lifetime, a process transits through several states. Queues in main memory keep track of the process by its process ID. A process resides on a queue according to its state; process states are defined in the `sys/proc.h` include file. Events, such as receipt of a signal, cause the process to transit from one state to another.

Processes can be described as being in one of the following states:

idle	Process has either just been `fork`ed; an idle process can be scheduled to run.
run	Process is on a run queue, available to execute in either kernel or user mode.
stopped	Executing process is stopped by a signal or parent process.
sleep	Process is not executing, while on a sleep queue (for example, awaiting I/O to complete).
zombie	Having exited, the process no longer exists, but leaves behind for the parent process some record of its execution.

These states and their transitions are shown in Figure 5-4.



LG200211_018

Figure 5-4. Process States and Transitions

5

When a program starts up a process, the kernel allocates a structure for it from the process table; the process is now in idle state, waiting for system resources. Once it acquires the resource, the process is linked onto a run queue and made runnable. When the process acquires a time-slice, it runs, switching as necessary between kernel mode and user mode. If a running process receives a `SIGSTOP` signal (as with control-Z in `vi`) or is being traced, it enters a stop state. On receiving a `SIGCONT` signal, the process returns to a run queue (in-core, runnable). If a running process must wait for a resource (such as a semaphore or completion of I/O), the process goes on a sleep queue (sleep state) until getting the resource, at which time the process wakes up and is put on a run queue (in-core, runnable). A sleeping process might also be swapped out, in which case, when it receives its resource (or wakeup signal) the process might be made runnable, but remain swapped out. The process is swapped in and is put on a run queue. Once a process ends, it exits into a zombie state.

Interrupts

Here is an example of how interrupts work:

Process A might make a system call requiring disk I/O. While waiting for the disk I/O to complete, Process A goes to sleep. Meanwhile Process B starts up. The disk I/O completion generates an interrupt. In the course of completing the disk I/O, the driver controlling it also puts the sleeping Process A onto a run queue, making it runnable at a high priority, because Process A now holds a resource. Interrupt activity "belongs" to Process A, but happened while Process B was running. The interrupt is asynchronous to the thread of execution.

Signals

Signals are transmissions between processes and are used for interprocess communication (IPC). Signals are also viewed as events to which processes respond.

An example of signal use is the `kill` command, which sends a `SIGTERM` signal to terminate the specifies processes.

When a process receives a signal, the process might ignore the signal, terminate, defer the signal, or execute a signal handler. HP-UX defines several signal interfaces that allow a process to specify the action taken upon receipt of a signal. (See *sigaction*(2) *signal*(2), *sigvector*(2), *bsdproc*(2), and *sigset*(2V) in the *HP-UX Reference Manual* for the various HP-UX signal interfaces. See *signal*(5) for description of HP-UX signals. Acceptable signal values are defined in `/usr/include/sys/signal.h`.)

Multiprocessing

Standard HP-UX executes in a uniprocessing system architecture consisting of one *central* processing unit (CPU), memory, and peripherals. The architecture of Series 870 and 890 systems supports **multiprocessing (MP)**—two, three, or four system processing units (SPUs), memory shared among the processors, and peripherals. Multiprocessing is a limited form of parallel processing.

Header files that define MP include `mp.h`, `spinlock.h`, and several semaphore-related header files. MP header files can be read but they define operating-system behavior that is not configurable.

How Multiprocessing Compares to Uniprocessing

Despite architectural differences, multiprocessing systems execute code identically to standard HP-UX. Although multiple processes can run concurrently on the several processors, all processors use the same kernel code.

Symmetry

Symmetry in MP has to do with use of the SPU. With some **asymmetrical** MP systems, only a primary SPU can execute I/O and system code; other SPUs execute only user code. With symmetrical MP, every SPU can execute I/O and system code. One copy of the kernel resides in memory and controls all processors. Since any kernel task can execute on any processor in the MP system, HP-UX MP is termed **symmetrical**.

Multiprocessor Boot-Up

Boot-up procedure in an MP system is almost identical to that of a uniprocessing system. At boot-up, the processor-dependent code (PDC) arbitrates among the multiple processors to determine the "monarch processor." The serf (non-monarch, or remaining) processors wait in "rendezvous code," while the monarch processor alone executes the bootup code. Upon reaching the MP portion of the configuration code (near the end of booting process), the monarch processor signals each serf processor in turn, to bring it into the multiprocessing environment, at which time they go into idle (see Figure 5-5, ready to run processes. Once all serfs are brought in, the monarch processor can go into idle also. From idle, processors do their work: they look in run queues for processes that need CPU time, grab locks, execute processes, release locks, and thus perform their normal operations on behalf of the kernel.

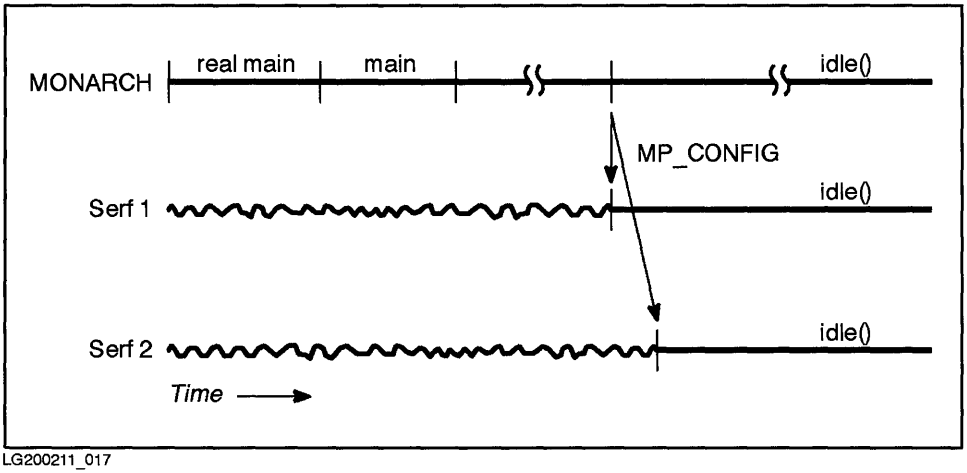


Figure 5-5. Multiprocessors Booting Up

Scheduling Processes using Semaphores and Spinlocks

When processes execute simultaneously, it becomes harder for the operating system to maintain the integrity of data. To solve this problem, HP-UX controls access to data by using **semaphores**.

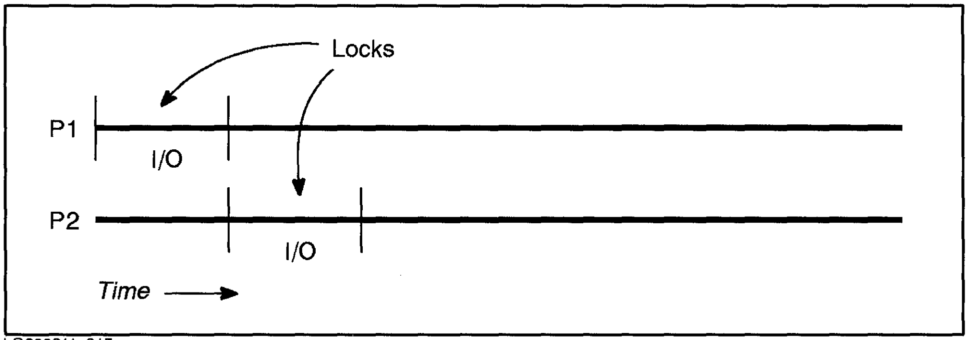
Semaphores (and spinlocks, a kind of semaphore) are a locking mechanism for ensuring that only one processor accesses critical kernel resources at a time. Critical kernel resources include run queues, file and inode tables, linked lists, any kind of table that processes might want to access or change.

A **spinlock** is held very briefly; other semaphores lock resources for longer durations and allow the processor to perform another task (such as switch processes) while the lock is held.

The MP system kernel uses semaphores to protect global data structures while multiple processors are executing concurrently. Locks are also used in a uniprocessing system to ensure data integrity.

Figure 5-6 shows in a very general way how semaphores and other locks allow processes to synchronize execution.

5



LG200211_015

Figure 5-6. Processes Lock Kernel Resources to Synchronize Execution

5

Processes run concurrently, but each lock may only be held by one process at a time. If another process attempts to acquire an already-held lock, the process must wait for the lock to be released.

The example shown in Figure 5-7 demonstrates how the kernel allocates critical regions to concurrently running processes. A process (P1), executing in user mode on SPU 1, might take a trap, which interrupts the process's execution and sends it into kernel mode, where it starts executing system code. P1 puts itself on a run queue, which pulls another process (P2) off the run queue. P2 starts to run. Meanwhile, a third process (P3), executing in user mode on SPU 2, takes a trap into kernel mode and wants to go on the run queue. Since P2 is executing in the critical region and controls the spinlock, P3 must wait, spinlocked, until P2 releases the lock; then P3 can continue to run.

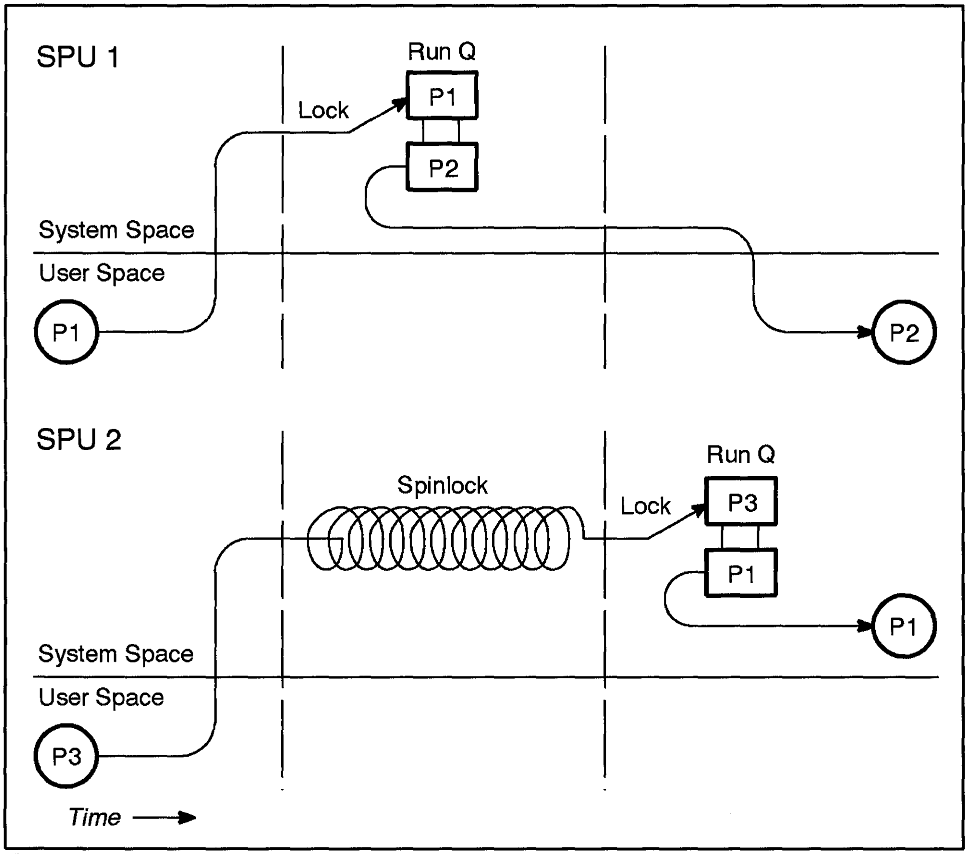


Figure 5-7.
Kernel Allocates Resources to Three Concurrent Processes on Two Processors Using Semaphores and Spinlocks

When multiple processes want to access critical resources at the same time, there is potential for a **race condition** or failure. Race conditions might occur any time a program takes an action dependent on data. Spinlocks enable the kernel to arbitrate between competing processes so that each process gets to access resources in a prioritized order.

5

For example, suppose there are two processes, executing on two SPUs, each process accessing the same location in kernel memory simultaneously (in lockstep). Each process is counting objects, with the statement

```
nobjects=nobjects+1;
```

The value of `nobjects` is the number of objects counted, and equals `10`.

If the same code is running simultaneously without spinlocks, the two processors independently compute and change the total twice to `11`. With spinlocks, the first process must complete and release its lock before the second process can compute. The total is thus incremented twice, first to `11`, then `12`.

Each processor uses spinlocks to control access to data structures on behalf of its executing process. Likewise, competing processors must wait to obtain a spinlock held by another processor, until the lock is available.

5

Processor Affinity

Processor affinity is the tendency of a process to execute from beginning to end on the same processor. The kernel is inclined to allocate processing time so that this happens, because a process running on the same processor executes faster.

The kernel, however, must also balance the workloads among the SPUs, and will move a process from one SPU to another if necessary to achieve the best overall system throughput.

Uniprocessor Emulation Issues

Underlying the design of HP-UX multiprocessing is the concept of uniprocessor emulation. In all user interactions, HP-UX on MP behaves exactly as if the operating system were running on a single processor. As noted earlier, widespread use of spinlocks require that device drivers, in I/O operations and in virtually all areas of program design, share kernel data structures sequentially to ensure their integrity. Because of MP's greater complexity, programmers must take care to write code more carefully: timing hazards not seen in uniprocessing can occur in an MP environment. Careful regression testing on MP computers is essential to screen for timing hazards.

A problem resulting from this sequential sharing of resources is characterized by excessive idle time. This situation can be detected with tools, such as *top*(1M), *sar*(1M), and the optional HP products *LaserRX* and *Glance/UX*. These tools, however, will not reveal why the problem occurs; thorough review of your application design will be necessary.

6

Run-Levels

This chapter discusses run-levels; specifically:

- The `init` process and the `/etc/inittab` file
- Run-level 2 (multi-user run-level)
- Run-level s (administration run-level)
- User-defined run-levels.

What Is a Run-Level?

A **run-level** is a system state in which a specific set of processes are allowed to run. This set of processes is defined in the `/etc/inittab` file for each run-level. You can define up to six run-levels (1-6). 6

Run levels `s` and `2` are predefined: `s` is the administration run-level and `2` is the normal operating run-level (users must provide user name to gain access). Most systems do not need to define additional run levels or use any run levels other than `s` and `2`.

Predefined Run-Levels

There are three pre-defined run-levels as shipped: run-level 2, run-level 0, and run-level s.

Run-Level 2

The run-level in which your system automatically boots is called the **initdefault** run-level. As shipped, run-level 2 is the initdefault run-level. On multi-user systems, it may be necessary to make certain, additions to run-level 2 (such as the addition of more `getty` entries to `/etc/inittab`). For more information, refer to Chapter 2 and the *System Administration Tasks* manual.

Run-Level 0

Run-level 0 is a special run-level reserved for system installation. Do not run in run-level 0.

Run-Level s

6 Run-level s is a special run-level reserved for system administration tasks. Shutting down the system for system administration tasks brings you to run-level s. Change to run-level s only by using the `shutdown` command (i.e., do not execute `init s`). For details on switching to run-level s, see the *System Administration Tasks* manual.

The init Process

The `init` process (`/etc/init`) is the first process HP-UX starts during system startup. The `init` process has process ID one (1) and has no parent.

The `init` process reads the `/etc/inittab` configuration file, which defines what processes to start at various run-levels. When `init` is run at system startup time, it typically brings the system to the **initdefault** run-level. An entry in `/etc/inittab` defines the initdefault run-level for your system. As shipped, the initdefault run-level is run-level 2 (multi-user HP-UX).

/etc/inittab

The `init` process reads the `/etc/inittab` file a line at a time. Each line contains an entry describing an action to take. Typically, entries tell `init` to run a process in a given run-level or run-levels. For example, the following entry tells `init` to run the `/etc/getty` process in run-levels 0 and 2 through 6:

```
co:023456:respawn:/etc/getty console H
```

The syntax of `inittab` entries is:

id: *run-levels*: *action*: *process*

id — A character id, from 1 to 4 characters long, which uniquely identifies the entry.

run-levels — Defines the run-level(s) in which the entry should be processed. Multiple run-levels can be specified. If this field is empty, then *all* run-levels are assumed.

action — Identifies what action to for this entry. The *actions* we discuss here are:

- `initdefault` — Causes the initial (default) run-level to be the value of the *run-levels* field. If more than one run-level is specified in *run-levels*, `init` uses the highest specified run-level.
- `boot` — Run the command specified in the *process* field at boot-time only. Do not wait for *process* to die before reading the next entry.
- `bootwait` — Run the command specified in the *process* field at boot-time only. Wait for *process* to die before reading the next entry.
- `wait` — On entering the run-level that matches the *run-levels* field of this entry, run *process* and wait for it to die before reading the next entry.
- `respawn` — On entering the run-level that matches the *run-levels* field of this entry, run *process* if it is not already running. Do not wait for *process* to die

before reading the next entry. When *process* dies, run it again.

process This is a shell command to be run, if the entry's *run-levels* matches the run-level and/or the *action* field indicates such action.

Comments of the form `# comment` can be appended to entries.

Setting the Default Run-Level

The following entry, which appears near the start of `/etc/inittab`, sets the default run-level:

```
is:2:initdefault:
```

This line tells `init` that the initdefault run-level is 2.

Before bringing the system to run-level 2, `init` processes all entries whose *action* field is `boot` or `bootwait`; such processes called **boot processes**. Two such entries are: `/etc/bcheckrc` and `/etc/brc`.

6

```
bl:bootwait:/etc/bcheckrc </dev/syscon >/dev/syscon 2>&1
bc:bootwait:/etc/brc 1>/dev/syscon 2>&1
```

These entries tell `init` to execute the `bcheckrc` and `brc` shell scripts, respectively. `init` must wait for each process to die before moving to the next entry. In addition, these processes are run only at boot-time.

After running the boot processes, `init` runs all processes at the initdefault run-level. Typically, these include `/etc/rc` and `/etc/getty`. For details on these processes, see Chapter 2.

User-Defined Run-Levels

Because of special requirements, some systems may need additional run-levels (other than 2 and s). Adding new run-levels involves changing **/etc/inittab**. For details on creating your own run-levels, see the *System Administration Tasks* manual.

Anyone having write permission to the file **/etc/inittab** can create new run-levels or redefine existing run-levels. Make sure that the permissions on **/etc/init** and **/etc/inittab** are:

```
-r-xr--r--    root    other              /etc/init
-rwxr-xr-x    root    root               /etc/inittab
```

Changing Run-Levels

When you start up your system, you will be in run-level 2. This means that at startup your system executed the **inittab** commands associated with run-level 2.

6

You can move freely between run-levels as long as entering the new run-level does not kill user or system processes that may have begun in the previous run-level. (See the *System Administration Tasks* manual for details on switching run-levels.) Two examples of possible problems are discussed below:

- If you change to a different run-level, the processes corresponding to entries in **inittab** that do not explicitly include the new run-level will automatically be terminated. For example, assume you are changing from run-level 2 to run-level 3. If the **getty** entry for the console does not have the action **respawn** for **run-levels** 3, entering run-level 3 from run-level 2 will cause the console to die.
- A user logs off prior to a change in run-level. You then change run-levels from run-level 2 to run-level 4. When the user attempts to log in, he cannot, unless his **/etc/getty** entry contains both run-level 2 and run-level 4.

7

Memory Management

Memory is high-speed data storage, implemented using various hardware devices on the HP-UX system. Each device stores and retrieves data.

The data and instructions of any process (a program in execution) must be available to the CPU by residing in **physical memory** at the time of execution. RAM, the actual **physical memory** (also called "main memory"), is shared by all processes. To execute a process, the kernel executes through a per-process **virtual address** space that has been mapped into physical memory.

The term "memory management" refers to the rules that govern physical and virtual memory and allow for efficient sharing of the system's resources by user and system processes.

Memory management allows the total size of user processes to exceed physical memory by using an approach termed **demand-paged virtual memory**. Virtual memory enables you to execute a process by bringing into main memory parts of the process only as needed, that is, on demand, and pushing out parts of a process that have not been recently used.

The system uses a combination of swapping and paging to manage virtual memory. **Swapping** involves moving entire processes between main memory and secondary memory (usually a mass storage device, such as a disk), whereas **paging** involves moving smaller units (called pages) between main memory and secondary memory.

This chapter provides an overview of HP-UX memory management, including:

- Physical and virtual memory
- Mapping files into virtual memory
- Shared libraries
- Swapping

Physical Memory

The physical memory of most interest to system administrators is the **random access memory (RAM)**. RAM usually consists of memory cards that plug into the computer's backplane. For the CPU to execute a process, the relevant parts of a process must exist in RAM.

The more main memory in the system, the more data it can access and the more (or larger) processes it can execute without having to page or swap. This is because the system can retain more processes in main memory, thus requiring the kernel to page less frequently. Memory-resident resources (such as page tables) also take up space in main memory, reducing the space available to applications.

During system startup, the system displays on the system console the amount of physical memory installed:

`real mem =` *no_of_bytes*

Refer to the HP *Configuration Guide* for your system's minimum, maximum, and recommended RAM requirements.

Power and Data Permanence

A characteristic of memory is its volatility or nonvolatility—whether or not a storage medium retains data when power is removed. Before the system boots, only data or microcode stored in nonvolatile memory (ROM, EPROM, magnetic tape, disk) is available.

Once the system is brought up, data is stored and used in RAM. Data held in RAM is volatile; that is, it is not retained in RAM when the system is brought down.

Transactions between RAM and Disk

At boot time, the system loads the operating system from disk. (This is essentially what booting means.) The operating system then resides in RAM until the system is shut down.

When the user runs programs and commands, they too are loaded from disk into RAM. When a program terminates, it is usually flushed out of RAM (that is, the operating system frees the memory used by the process).

Complex swapping and paging algorithms determine when data and code for currently running programs will get returned from RAM to disk.

User and system programs write data to disk (for example, to update the password file or write a database record). This data-to-be-written is either written directly to RAM (if raw data) or buffered in cache and written to disk in relatively big chunks. Programs also read files and database structures from disk into RAM. Buffering algorithms try to minimize disk access by going to disk as infrequently as possible; disk access is a bottleneck on all systems.

When you issue the `sync` command before shutting down a system, all modified buffers of the buffer cache are flushed (written) out to disk.

On the Series 800, if the system loses power for a short time and powerfail is enabled, the powerfail routines will put the system in a consistent state and bring it back up without the user having to reboot it.

There are two other characteristics of physical memory involved at system start-up: available memory and lockable memory.

Available Memory

Not all physical memory is available to user processes. The kernel (`/hp-ux`) always resides in main memory (that is, it is never swapped), occupying approximately 1.6 MB on a Series 300/400 and 3 MB on a Series 700/800 system. (Note, however, these are static sizes only, and do not include kernel tables, daemons, device drivers, processes, diagnostics, user interfaces, or other executing code on a working system.)

The amount of main memory not reserved for the kernel is termed **available memory**. Available memory is used by the system for executing user processes.

Instead of allocating all its data structures at system initialization, the HP-UX kernel dynamically allocates and releases some kernel structures as needed by the system during normal operation. This allocation comes from the available memory pool; thus, at any given time, part of the available memory is used by the kernel and the remainder is available for user programs.

During system startup, the system displays on the system console the amount of available memory:

`avail mem =` *no_of_bytes*

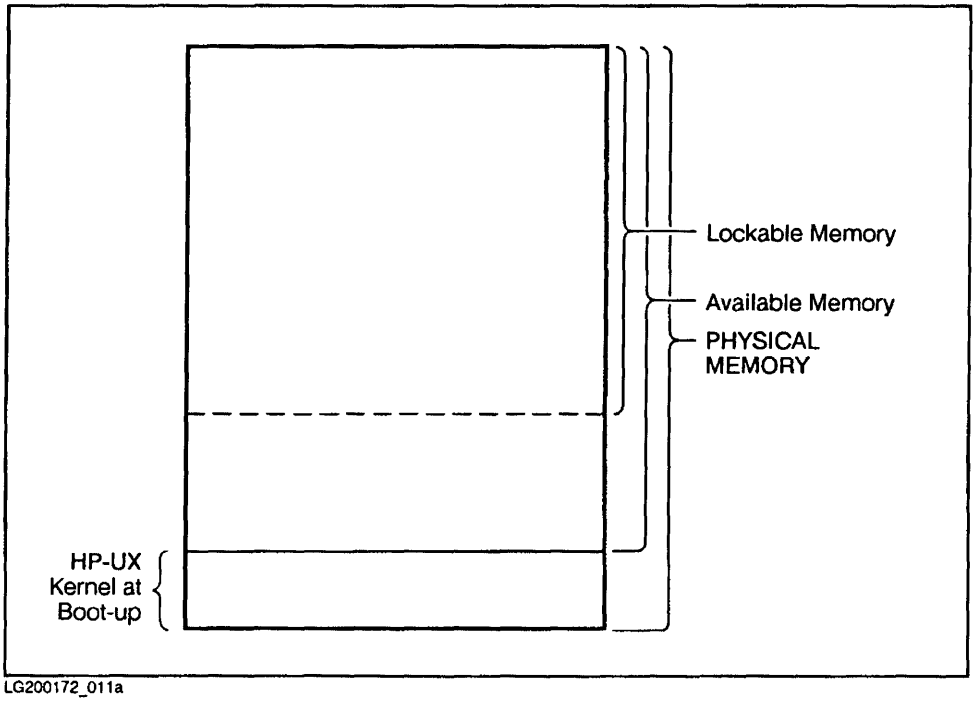


Figure 7-1. Physical Memory Available to Processes

Lockable Memory

As shown in Figure 7-1, physical memory that can be "locked" (that is, its pages kept in memory for the lifetime of a process) by the kernel or `plock()` is known as **lockable memory**.

User processes can lock memory using the `plock` or `shmctl` system call (see *plock*(2) or *shmctl*(2) in the *HP-UX Reference*). Locked memory cannot be paged or swapped out. Typically, locked memory holds frequently accessed programs or data structures, such as critical sections of application code. Keeping them memory-resident improves system performance.

By default, lockable memory can be no more than three-fourths of available memory during bootup. Available memory is a portion of physical memory, minus the amount of space required for the kernel and its data structures. The size of the kernel varies depending on the number of interface cards, users, and values of the tunable parameters; thus, available memory varies from system to system.

Lockable memory is extensively used in "real-time" environments, like hospitals, where some processes require immediate response and must be constantly available.

A Series 800 feature called pseudo-swap reservation (discussed at the end of this chapter) also does not allow lockable memory to exceed three-fourths of available memory.

If the default value of lockable memory is changed, care must be taken to allow sufficient space for paging and swapping. As this space is decreased, the system is more likely to deadlock. The system parameter to change the amount of unlockable memory is `unlockable_mem`. (See *System Administration Tasks* manual for information on tunable operating-system parameters.) 7

During system startup, the system displays on the system console the amount of its lockable memory (along with available memory and physical memory). You can display the values later by running `/etc/dmesg`.

`lockable mem =` *no_of_bytes*

`lockable_mem` is the total amount of physical memory all users can lock down at once. This value can be changed by modifying the operating-system parameter, `unlockable_mem`.

The system has boundary conditions requiring that `unlockable_mem` be more than zero and less than the amount of memory available after boot. If `unlockable_mem` is set to zero or a negative value, the kernel will compute an appropriate default value at boot time.

Available memory minus the memory locked by processes is the memory actually available for virtual memory demand paging.

You can determine the default value of `unlockable_mem` by subtracting the amount of lockable memory from the available memory during boot-up.

```
available memory - lockable memory = unlockable_mem
```

The resulting value must be less than three quarters of free memory.

For example, the following boot information was displayed on a Model 835:

```
physical page size = 2 KB
avail mem = 9670656 bytes
real mem  = 1677216 bytes
lockable mem  = 659456 bytes
```

The system's unlockable memory would be

```
9670656 - 659456 = 9011200 bytes
```

For details on setting lockable memory, refer to the `unlockable_mem` tunable parameter in the system parameters chapter of the *System Administration Tasks* manual.

7

Secondary Storage

Main memory (RAM) stores computer data required for program execution. During process execution, data resides in two faster implementations of memory found in the processor subsystem, registers and cache. Program files are kept in **secondary storage** or **secondary memory**, typically disks accessible either via system buses or network. Data is also stored when no longer needed in main memory, to make room for active processes.

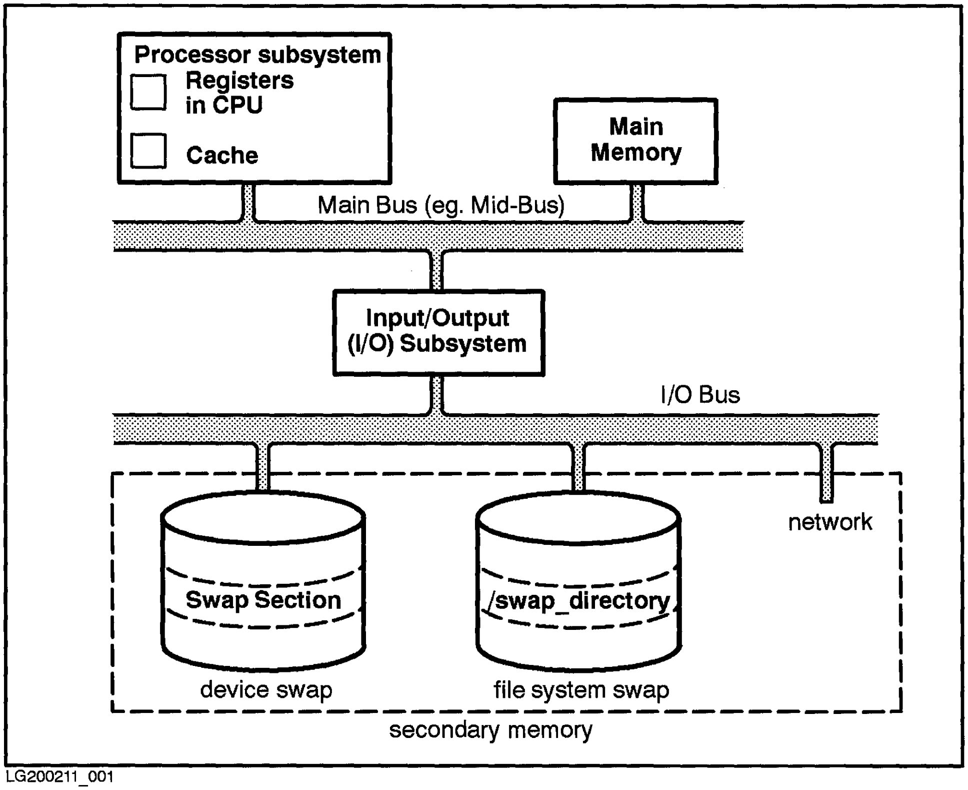


Figure 7-2. Relationship between Main Memory and Secondary Storage

A transitory form of secondary data storage is termed **swap**, dating from early UNIX implementations that managed physical memory resources by moving entire processes between main memory and secondary storage. Besides swapping, HP-UX uses paging, a more effiicent memory resource management mechanism for virtual memory.

While executing a program, data and instructions can be swapped (copied) to and from secondary storage if the system load warrants such behavior.

Swap space is initially allocated when the system is configured. **Device swap** can take the following forms:

- an entire disk (Series 300/400/700/800)
- a designated area on a disk (Series 300/400/700)

- a section or LVM logical volume of a disk (Series 800)
- a software disk-striped partition on a disk (Series 700); see *sdsadmin*(1M)

You can also configure a file system so that remaining space not used for files is used for swap; this is termed **file-system swap**. If more swap space is required, it can be added dynamically to a running system, as either device swap or file-system swap.

The `swapon` command is used to allocate disk space or a directory in a file system for swap.

Note You cannot remove swap without rebooting the system.

The concepts of swap and swap-space management are further discussed in "Swap Space Management," later in this chapter. For procedures on allocating swap space, see the *HP-UX System Administration Tasks* manual for your system, Chapter 7, "Managing Swap Space" and Appendix B, "Swap Space Computation".

HP-UX Demand-Paged Virtual Memory

Early UNIX systems transferred only entire processes between the swap device and main memory while executing a program. Complex algorithms governed the priority by which systems moved processes to and from swap. To better accommodate larger programs, HP-UX memory-handling algorithms can also allow **pages** (individual units of memory) of a process to be read (paged) in and out of memory.

Hardware Perspective on Memory Transactions

Most of the kernel's memory-management code operates independently of hardware, but some code is specific to the Series 300/400 or 700/800 hardware.

When a program is compiled, the compiler generates addresses, called **virtual addresses**, for the code. These virtual addresses must be mapped into memory to a physical address for the compiled code to execute. User programs use virtual addresses only. The kernel and the hardware coordinate a mapping of these virtual and physical addresses for the CPU, called "address translation." Figure 7-3 shows how the system uses the addresses to locate the process is in memory:

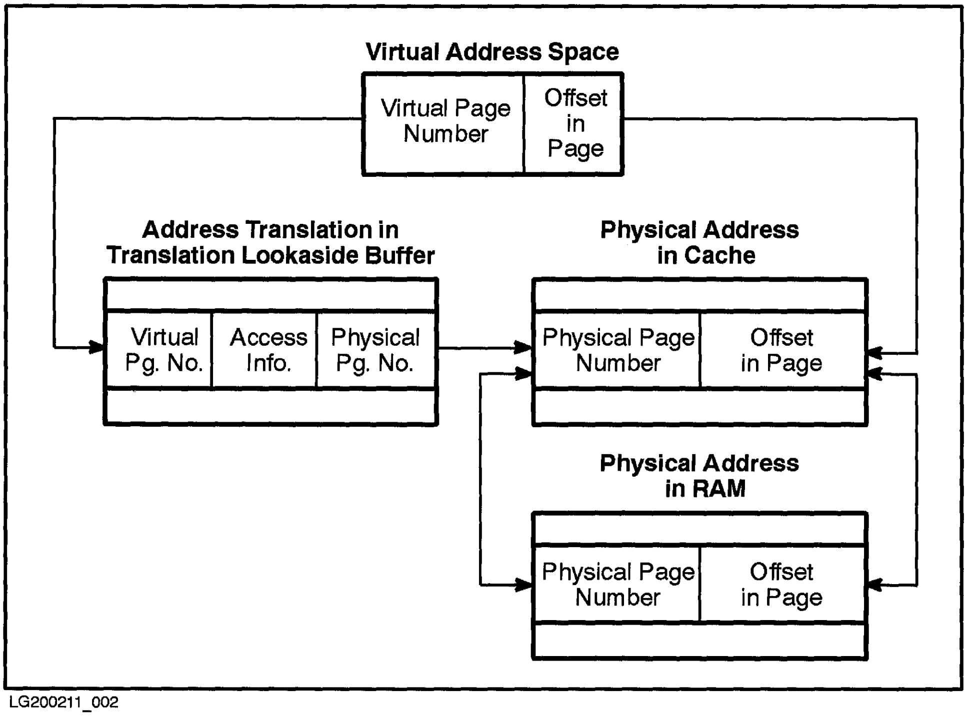


Figure 7-3. Virtual Address Translation

When a process executes, it stores its code (text) and data in processor registers for referencing. If the data or code is not present in the registers, the processor goes out to the cache, a small high-speed memory between RAM and the processor, to fill a cache line. If the data is not in cache, the processor consults the translation tables in RAM that hold the mapping between virtual and physical addresses of the data. If not in RAM, the data and code might have to be paged into RAM from disk, in which case the disk-to-memory transaction must be performed.

A **memory management unit** (MMU) manages the interface between blocks of virtual address spaces and their physical memory locations. The MMU also ensures that processes do not illegally access each others' address space. Access to data in memory is thus facilitated and protected at a page-by-page level.

An important element of the memory management unit is the **translation lookaside buffer (TLB)** hardware in the processor. The TLB caches the most recently used virtual-to-physical address translations with their corresponding access information. Once an address is translated, it can be used to reference physical memory.

It is also interesting to compare the relative magnitude of these memory components. The virtual address space can potentially be a thousand times greater than the physical address space, while the physical address space might be a thousand times greater than the TLB. These relationships enable the CPU to execute programs much larger than the available physical memory. It also lets you run many more programs at a time than you could without a virtual memory system.

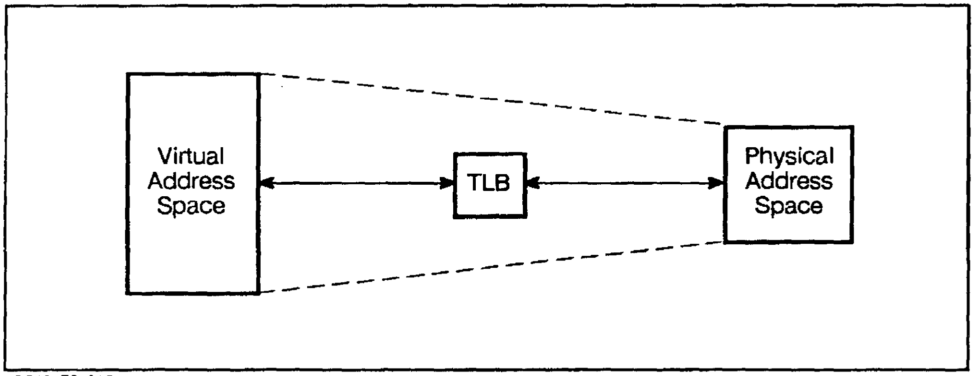


Figure 7-4. Role and Magnitude of the Address-Translation Components

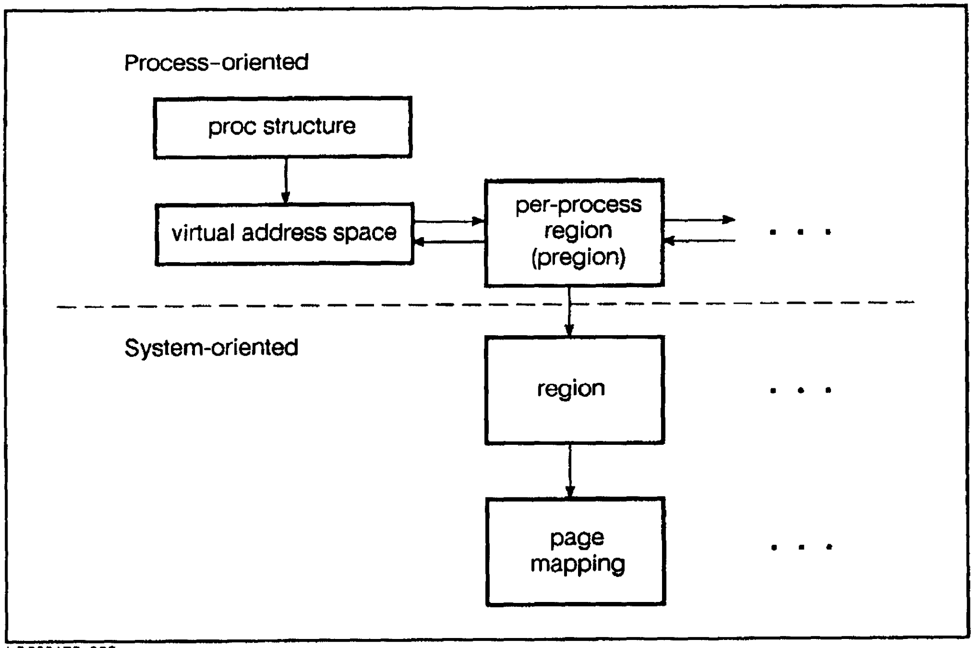
7

Process-to-Page Mapping

When a process executes, various dynamic structures are used to locate the memory segments, translate them from virtual to physical addresses, and manipulate them to achieve the results desired. Figure 7-5 shows how these structures point from a process to its segment page mappings.

When a process executes, the kernel maintains a `proc` structure, a data structure for each active process. The `proc` structure points to a virtual address space, which forms the header of a doubly linked list of several per-process regions, each corresponding to some element of the process (such as code, data, `bss`, shared memory). In turn, each per-process region points to a region, which points to page mappings of the code in physical memory.

Subsequent sections of this chapter describe the virtual address space and its related structures.



LG200172_026

Figure 7-5. Simplification of an Active Process Executing in Main Memory.

Pages

Pages are the smallest contiguous block of physical memory that can be allocated for storing data and processes. Pages are also the smallest unit of memory protection. As of the current release, the page size of all HP-UX systems is 4 KB.

A free page is a page of physical memory not mapped to a virtual address; that is, not currently being used by any process. A used page is mapped to a virtual address. The system keeps track of all free and used pages in physical memory.

Every physical page in the system is represented by an entry in the kernel's `pfdat` array data structure. A `pfdat` entry keeps information for the hardware-independent code on page conditions such as a page's validity, the state of the page, and whether the page has been modified. Implementation of the actual page mapping is hardware dependent.

You can lock critical pages of processes in memory to prevent their being paged out by using the system call *plock*(2) as described in the *HP-UX Reference*; see "Lockable Memory", earlier in this chapter.

Virtual Address Space

Virtual memory uses a structure for mapping processes termed the **virtual address space**, described in the kernel header structure `/usr/include/sys/vas.h`. The virtual address space contains information and pointers to the memory that the process can reference.

One virtual address space exists per process and serves several purposes:

- It provides the overall description of each process.
- It contains pointers to another element in the memory-management subsystem—per-process regions, or `pregions`.
- It keeps track of `pregions` most recently involved in page faults (explained in "How the Kernel Executes Processes using Demand Paging").
- It contains pointers to page tables that enable the CPU to access physical addresses of data. (Series 300/400 only).

On HP-UX systems, the entire virtual-memory system can handle addresses of 32 bits, allowing processes a maximum virtual address space of 4 Gigabytes.

The Series 300/400 has a linear address space without base and offset. All memory is conceptually contiguous. Each process (including the kernel) shares the entire 2^{32} (4 GB) address range.

On the Series 700/800, space registers hold either 16 bits (for 48-bit addressing) or 32 bits (for 64-bit addressing) and are used to point to the virtual space to be accessed. The specific location within that space is specified by a 32-bit quantity called the **byte offset**.

The present Series 700/800 48-bit addressing implementation can be thought of as having 2^{16} 32-bit spaces.

Configuring Virtual Address Space

Operating-system parameters limit the size of the code, data, stack, and shared-memory segments for each individual process. These parameters have predefined defaults, but you can reconfigure them in the kernel using the procedures outlined in the *System Administration Tasks* manual. The following list shows configurable system parameters for code, data, stack, and shared memory:

`maxtsiz`	Limits the size of the code (text) segment.
`maxdsiz`	Limits the size of the data segment.
`maxssiz`	Limits the size of the stack segment.
`shmseg`	Limits the number of shared memory segments that can be attached to a process.
`shmmni`	Limits the number of shared-memory identifiers.

7

Per-Process Regions (pregions)

The virtual address space structure points to **per-process regions**, or **pregions**. `Pregions` are logical segments that point to specific segments of a process, including code (text, or process instructions), data, `u_area` and kernel stack, user stack, shared-memory segments, and shared-library code and data segments.

`Pregions` hold page protections and the number of pages mapped to each segment.

Each segment also corresponds to the segments of virtual address space illustrated in Figure 7-6, Figure 7-7, and Figure 7-8.

The **text segment** (or **code segment**) holds a process's executable object code and may be shared by multiple processes. The maximum size of the text (or code) segment can be changed by the configurable operating-system parameter `maxtsiz`.

The **data segment** contains a process's initialized and uninitialized data structures (`bss`). It can grow as needed by a program's run-time logic (using *sbrk*(2), *malloc*(3C), or *malloc*(3X), as described in the *HP-UX Reference*). The total allotment for initialized data, uninitialized data and dynamically allocated memory can be changed by the configurable operating-system parameter `maxdsiz`.

The **u_area** contains information about process characteristics. The **kernel stack segment**, which is in the `u_area` segment, contains a process's run-time stack during kernel mode. Both `u_area` and kernel stack segment are fixed in size.

The **user stack segment** contains a process's run-time stack during user mode. The user stack expands during process execution; its default maximum size can be changed using the configurable operating-system parameter `maxssiz`.

Shared memory segments are typically used when multiple processes must share data (for example, in a windowing system, where all window processes must be able to update common data structures). They can be created using the *shmop*(2) system call (see specifications in the *HP-UX Reference*).

Each **shared library segment** typically has three `pregions`—code, initialized data, and `bss`.

7

I/O mappings are the ranges of addresses the operating system uses to deal with I/O devices.

When a process executes, the memory management subsystem traverses the `pregions` to find pointers to another data structure, called a **region**, which points to the address of the desired page of code or data.

Per-Process Region Layouts

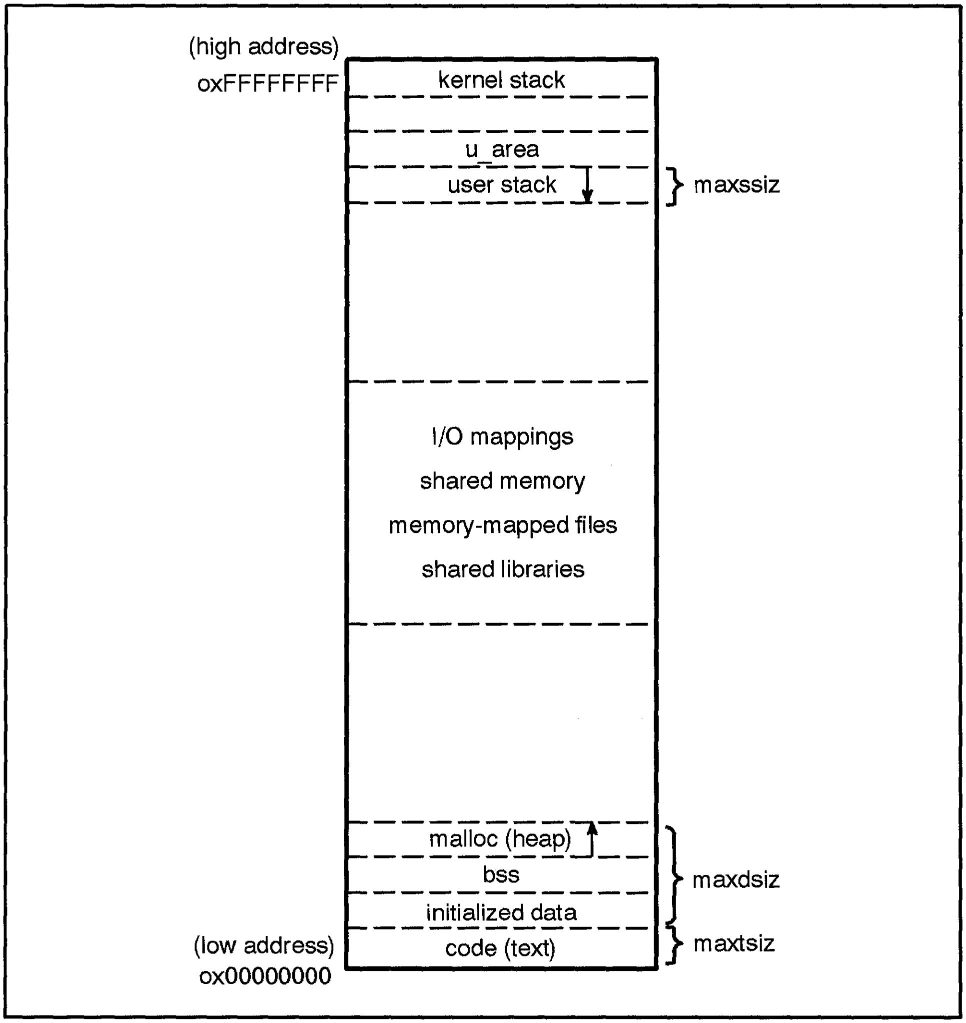
Per-process regions (`pregions`) represent the virtual address space in memory.

The layouts of the Series 300/400 and 700/800 virtual address space differ in specifics. This section shows the mappings of both Series 300/400 and Series 700/800.

Placement of Segments in Address Space (Series 300/400)

The segments of virtual address space for a process reside in virtual memory as shown in Figure 7-6. In the Series 300/400 memory design, the percentage and boundaries of memory address are not critical.

As a process is executed, the stack segments grow in the direction shown by the arrows in Figure 7-6. Segments are arranged to allow for ample memory allocation, to prevent overlap that would cause error (such as ENOMEM) and kill the process.



LG200211_003

Figure 7-6. Series 300/400 User Process Virtual Address Space

7

Placement of Segments in Address Space (Series 700/800)

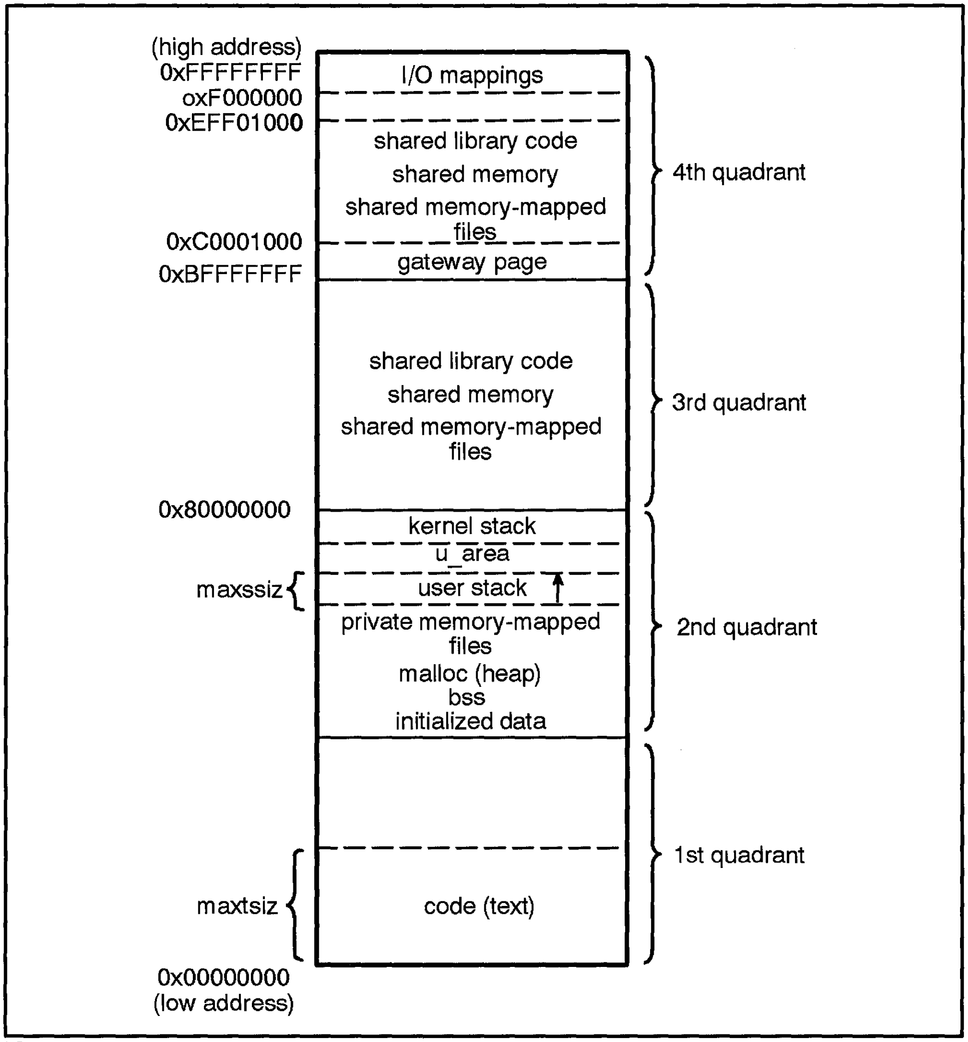
In the Series 700/800, the virtual address space is divided into one-GB quadrants and addressed by units of 32 bits. Each quadrant has several segments associated with it, as shown in Figure 7-8 and explained on the next page.

The basic segments (quadrants) of the Series 700/800 can be described as follows:

- First one-GB quadrant always contains the process's code and sometimes some of the data.
  Text is addressed from 0x00000000 to 0x3FFFFFFF.
- Second quadrant contains data (static data, stack, and heap).
  Data is addressed from 0x40000000 to 0x7FFFFFFF.
- Third quadrant, which is addressed from 0x80000000 to 0xBFFFFFFF, is used somewhat differently on Series 700 and Series 800:
  - On the Series 700, the third contains shared memory, shared mapped files, and shared library code.
  - On the Series 800, the third quadrant contains memory-mapped segments (such as shared-library code).
- Fourth quadrant, which is addressed from 0xC0000000 to 0xFFFFFFFF, contains shared-memory segments. On the Series 700, fourth quadrant also contains shared memory-mapped files and shared library code. On PA-RISC architecture (both Series 700 and 800), addresses from 0xF0000000 to 0xFFFFFFFF are used for I/O space.

7

On the Series 700, an EXEC_MAGIC user executable (`a.out`) format allows data to start immediately after the code area in the first quadrant, instead of at the beginning of the second quadrant, and grow to the beginning of the user stack. Executables created with this format can handle more than 1 GB of data; refer to *ld*(1) in the *HP-UX Reference Manual* for information.



LG200211_004

Figure 7-7.
Series 700 User Process Virtual Address Space with EXEC_MAGIC implemented.

7

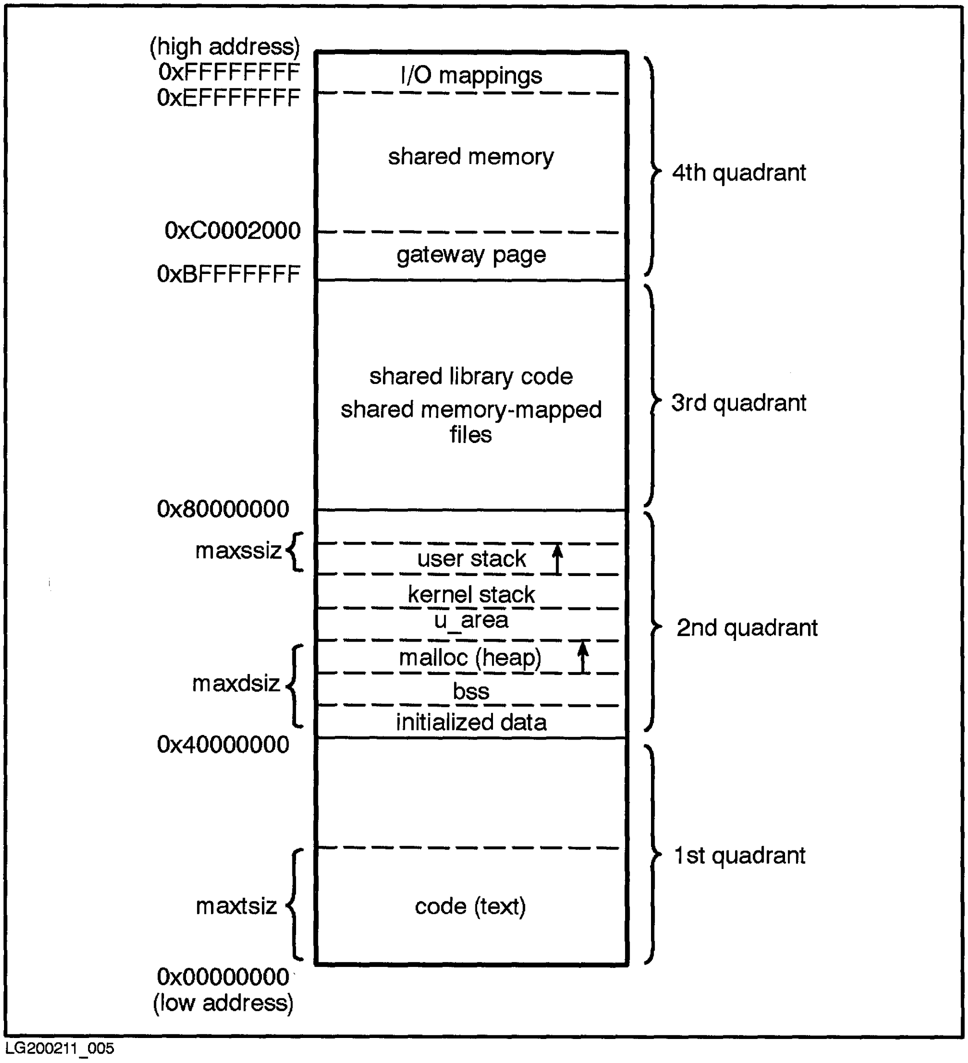


Figure 7-8. Series 800 User Process Virtual Address Space

7

Regions

Regions are data structures unique to the file or chunk of memory being accessed. Regions represent ranges of addresses and inform the process about where the data exists in physical memory.

Each process segment's per-process region maps directly to a segment of memory in a region. Regions are associated with the system rather than a process.

In turn, the region points to two kernel lists, which keep track of pages of data.

A region might be private or shared. A private region (such as a process's data segment, `u_area`, or stack) has only one `pregion` pointing to it. A shared region (such as shared memory, code, graphics) can have more than one `pregion` point to it, because more than one process might be accessing the same segment of memory.

Virtual Nodes (vnodes)

A region usually references a **virtual node (vnode)**, an interface for reading and writing pages of data between memory and disk. `Vnodes` are supersets of inodes that the MMU stores to keep track of swapped or paged memory while it is out on disk. `Vnodes` are categorized by file type—`hfs`, `nfs`, `dux`, `cdfs`, and `swap`—and are defined in the header file, `vnode.h`. Kernel routines read the stored file type when:

- Bringing requested and additional pages in from disk.
- Writing pages out to disk.
- Giving page information to the kernel for later computation.
- Duplicating page information for use by multiple processes.

Memory-Mapped Files (Series 300/400/700)

Memory-mapped files allow applications to map file data into a process's virtual address space. Once mapped, the process can directly manipulate the file data as a portion of memory. File data can be shared between processes more efficiently, resulting in higher I/O throughput for processes. Memory-mapped files are always page aligned. I/O to the files can be performed through direct loads and stores of memory instead of using *read*(2) and *write*(2), avoiding copying of data between user and system buffers. Memory-mapped files provide identical representation of in-memory and on-disk data. The data can be selectively synchronized to flush out a portion of a file range.

Files can be mapped into memory as either private or shared. Modifications to a private mapping are not visible to other processes, regardless of whether the other processes have created a private or shared mapping, or perform a `read` or `write` of the same file. The modifications are never written back to the file. Modifications to a shared mapping *are* visible to all other processes with a shared mapping and the modifications are written back to the disk file.

Memory-mapped files are well suited to randomly dispersed I/O, such as in database applications with data that is most frequently read, and which require frequent, small, random updates. Likewise, multiple processes that read huge data files can use memory-mapping for repeatedly searching and scanning the same data.

Several system calls manipulate memory mappings to a file. Also, user-level semaphores interface with the memory-mapped regions:

`mmap`	Create a mapping from a range in the process's address space into a range in a file.
`mprotect`	Modify the protections on memory pages mapped to a file.
`munmap`	Unmap a mapping created by `mmap`.
`msync`	Synchronize memory pages of a mapped file back to disk.
`msem_init`	Initializes a semaphore in a memory-mapped region.
`msem_lock`	Locks a semaphore in a memory-mapped region.
`msem_unlock`	Unlocks a semaphore in a memory-mapped region.
`msem_remove`	Removes a semaphore in a memory-mapped region.

For detailed information on these system calls, refer to section two of the *HP-UX Reference Manual*.

Limitations to Memory Mapping

Memory-mapped files cannot be locked into memory. Extensive use of memory-mapped files (`mmap`) might result in some degradation of performance.

Access to the same file by `read` and `write` system calls, on the one hand, and memory mapping, on the other hand, might produce inconsistent data, since both approaches maintain separate in-memory caches of file data. Use one access method exclusively for any given application.

The `mmap` interface does not map character device files and device-specific addresses (such as graphics framebuffers). Use *graphics*(7) for graphic devices and *iomap*(7) for other devices on the Series 300/400; use *framebuf*(7) for graphics devices on the Series 300/400/700/800.

The Series 700 memory-mapped file interface is defined to work in a 32-bit process address and cannot share data with processes outside this address space. The following limits apply to the Series 700 implementation:

- 0.9 GB is the maximum size of a single file that can be mapped entirely into memory.
- 1.75 GB (minus code size and data size) is the maximum size of any EXEC_MAGIC executable on a Series 700.
- 1.75 GB is the maximum combined size of all files mapped shared and shared memory, by all processes on a system.
- Only one fixed, contiguous mapping can exist of a shared mapped file in the shared virtual address space.

How the Kernel Executes Processes using Demand Paging

A compiled program has a header containing information on the size of the data and code regions.

As a process is created from the compiled code (by `fork` and `exec`), the kernel sets up its data structures and the process starts executing its instructions from user mode.

A **page fault** occurs when the process tries to access an address that is not currently in main memory.

The kernel switches execution from user mode to kernel mode and tries to resolve the page fault by locating the `pregion` containing the sought-after virtual address. The kernel then uses the `pregion`'s offset and region to locate information needed for reading in the page.

In main memory, the kernel also looks for a free page in which to load the requested page. If no free page is available, the system swaps or pages out selected used pages to make room for the requested page.

The kernel then retrieves (pages in) the required page from file space on disk. It also often pages in additional (adjacent) pages that the process might need.

Then the kernel sets up the page's permissions and protections, and exits back to user mode. The process executes the instruction again, this time finding the page and continuing to execute.

Pages are not loaded in memory until they are "demanded" by a process—hence the term, **demand paging**.

copy-on-write

An HP-UX enhancement to earlier UNIX implementations is the technique of "copy-on-write." The system used to copy the entire data segment of a process every time the process `fork`ed, increasing `fork` time as the size of the data and code segments increased.

On the Series 300/400, HP-UX implements **copy-on-write**, which enables the system to create processes more quickly. When a process forks, the parent process makes a duplicate image of itself at child virtual addresses. But instead of actually copying the pages, the parent marks them "copy-on-write." The pages continue to reside in the parent address space, and are not copied until the process actually writes on the page. Then, the old page is copied to a new page, giving the process access to the new page. More specifically, data pages being copied-on-write are shared until written, while for code pages, a private copy is created on write.

On Series 700/800 systems, HP-UX implements a variation called **copy-on-access**. Only one translation of a physical page is maintained; the parent process can point to and read a physical page, but copies it only when writing on the page. The child process does not have a page translation and must copy the page for either read or write access.

Maintaining Page Availability—vhand and swapper Daemons

Two **daemons** (background processes)—`vhand` and `swapper`—are involved in paging; the actual paging being performed by `vhand`, which is also known as the pageout daemon.

`vhand` monitors free pages and tries to keep their number above a threshold (shown in Figure 7-10). This ensures sufficient memory for the most efficient operation of demand paging.

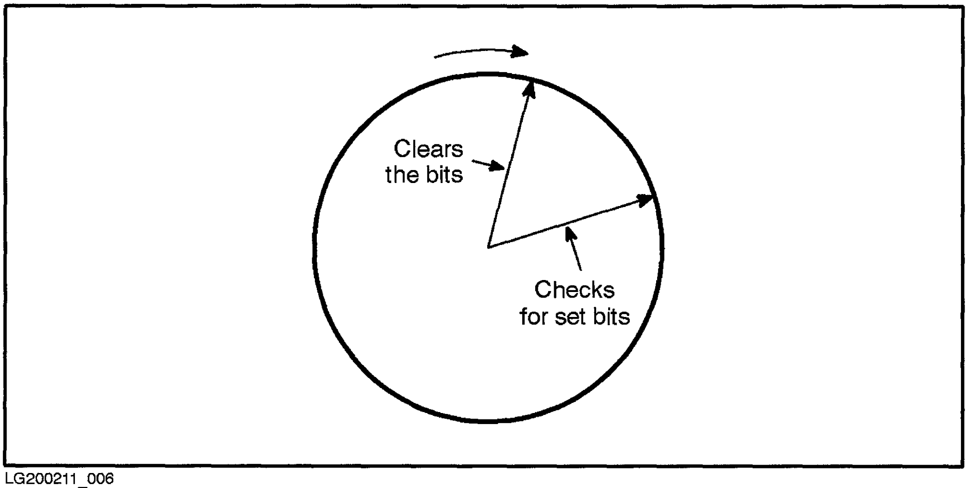


Figure 7-9. The Pageout Daemon `vhand`

As shown in Figure 7-9 the pageout daemon is like a "two-handed" clock, with the distances between the hands representing the size of main memory available to user processes. The first hand clears reference bits on a group of pages in an active pregion. If the bits are still clear by the time the second hand reaches them, the pages are paged out.

On the Series 800, the distance between the two hands is governed by a factory-set parameter called **vhandsk**, or "**vhand skew**." The larger the parameter, the longer pages can reside in main memory without being paged out.

On the Series 300/400/700, the kernel automatically keeps an appropriate distance between the hands.

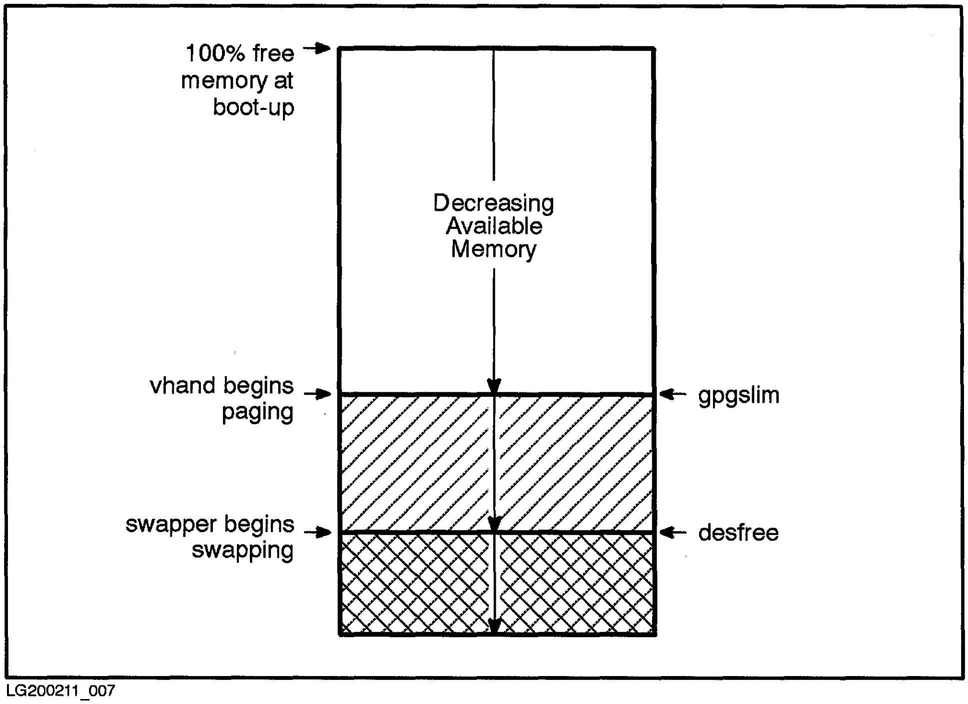


Figure 7-10. Maintaining Memory Availability

vhand decides when to start paging by determining how much free memory is available.

At the upper limit of the value `gpgslim`, paging occurs. The `swapper` daemon's upper limit is `desfree`. If free memory falls below `desfree`, the `swapper` detects the condition and becomes active also. The `swapper` deactivates processes and prevents them from running, thus reducing the rate at which new pages are accessed. Once `swapper` detects that available memory has risen above `desfree`, `swapper` reactivates the deactivated processes and continues monitoring memory availability.

Thrashing

On systems with very demanding memory needs (for example, systems that run many large processes), the paging daemons can become so busy swapping pages in and out that the system spends too much time swapping and not enough time running processes.

When this happens, system performance degrades rapidly, sometimes to such a degree that nothing seems to be happening. At this point, the system is said to be **thrashing**, because it is doing more overhead than productive work.

Consider, for example, that your system might be thrashing because it is handling a lot of disk activity. Perhaps a data base is in constant use, yet on the same disk is swap space. (On the Series 800, you can check system activity using the *sar*(1) command; see the manual page in the *HP-UX Reference*.) If so, you can minimize disk activity by moving a busy file system to a different disk, so that you are distributing busy disk activity among different spindles.

In many cases, even this approach is insufficient. Thrashing is often eliminated by adding more main memory to the system, thus alleviating the need to swap. This reduces the amount of time the system spends paging and swapping.

How the Memory-Management System Handles Executable Code

The speed at which processes execute is related partly to how the operating system accesses virtual address space segments of the compiled and linked object code (`a.out` or other executables).

If you are doing applications or system programming, you may be aware that a program's magic number and internal attributes determine which type of executable code—standard, shared, and demand loaded—are possible. (A later subsection, "Benefits and Shortcomings of Shared and Demand-Loaded Code," discusses this in more practical terms. Also see *magic*(4).)

The following *HP-UX Reference* manual pages describe magic numbers and internal attributes in detail:

- The *chatr*(1) command is used to change a program's internal attributes to shared or demand-loaded.
- The link editor *ld*(1) produces executable files from one or more object files or libraries.
- *magic*(4) describes predefined file types and magic numbers for HP-UX implementations.
- *a.out*(4) and its Series 300/400- and 700/800-specific manual pages describe the output file format from the assembler (*as*(1)), compilers, and link editor.

Series 300/400/700 and Series 800 computers have different capabilities. On the Series 300/400/700 computers, executable code can be either standard, shared or demand-loaded. Series 800 computers do not support standard executable code, but code can be shared or demand-loaded.

Table 7-1 and subsequent sections describe the types of executable code by several criteria:

- Addressing in separate segments of code and data.
- Capability of virtual address code segments being shared among multiple processes.
- Alignment of pages to corresponding address boundaries on both disk and in main memory, for direct memory-to-memory copy. (Object code should align on 4K page boundaries for all HP-UX systems.)

Although page size was 2 KB for some previous Series 800 releases, executables generated during these releases will be handled properly by the current release, but with some performance penalty. You might want to recompile for better performance.

Table 7-1. Characteristic Types of Executable Code

Type of Executable Code	Computer Architectures		
	Series 300/400	Series 700/800	Both
EXEC_MAGIC	- single segment (code and data) for read and write - no restrictions on alignment[1] - no sharing - code and data faulted in as needed	- EXEC_MAGIC is supported on Series 700, unsupported on Series 800.[2] - 4 KB-page aligned on disk	
SHARE_MAGIC	- not necessarily page-aligned on disk[1]	- 4KB-page aligned on disk[3]	- separate segments for code and data - code is read-only - code is shared - page aligned in memory - code loaded into memory entirely at execution - code and data faulted in as needed
DEMAND_MAGIC		- 4KB-page aligned on disk	- separate segments for code and data - code is read-only - code is shared - page aligned in memory - page aligned in disk[2] - code and data faulted in as needed

1 Executable code that is not page aligned on disk might suffer performance degradation, since it may be faulted in by a less optimal path.

2 Series 700 EXEC_MAGIC executables will not run on Series 800.

3 Programs built on the Series 800 can be executed on the Series 700; however, footnote 1 applies on a 4 KB-page size Series 700 for such executables. Note too, executables generated on previous Series 800 operating systems featuring 2K page size will run, but with some performance penalty. You might want to recompile for better performance.

Standard Executable Code (EXEC_MAGIC)

Note EXEC_MAGIC is not supported on the Series 800.

In earlier implementations of UNIX, compiled object files consisted of code (machine instructions) and data that occupied the same area of memory, with read, write, and execute permissions.

Under those circumstances, each time a common program like `vi`, for example, was run, a distinct copy of its code was read into main memory. The code was not shared, even when several `vi` processes were run simultaneously.

Because the `vi` code occupied area segments with write permissions, it was vulnerable to being overwritten. File header, code, data, and debugger code were not aligned in main memory with their corresponding page boundaries on disk, and so all code had to be read through a buffer cache before being copied to the user's address space. This slowed the translation.

The EXEC_MAGIC user-executable (`a.out`) format on the Series 700 allows creation of processes with more than 1 GB of data. This expanded data area is implemented with a new option to the linker; refer to *ld*(1) in the *HP-UX Reference Manual* for information.

Several factors contribute to the efficiency of EXEC_MAGIC on Series 700: Pages are aligned at 4 KB, improving address translation. Text for processes greater than 1 GB is relatively small, compared to data; thus, private code does not pose a problem. Also, when processes are greater than 1 GB, normally only one of them is run at a time.

Shared Code (SHARE_MAGIC)

Sharing of code segments reduces the amount of code kept in main memory. With **shared code** (also known as **shared code**), the code and data segments are separated, so that code can be read-only and data can have write permission. This protects code from being overwritten.

When several processes run the same program simultaneously, the processes use the same copy of code in main memory via pointers to the code's virtual address space. Only one copy of the code exists in memory, regardless of how many processes run the program. The system keeps track of multiple processes sharing code by maintaining a **use count**. These mechanisms decrease dramatically the amount of memory required for each user's process space.

For example, when a shared program such as `vi` is first loaded into the user code area, the use count for the program is set to one. While the first process is executing `vi`, if another process invokes `vi` also, no additional memory is allocated because the code already resides in main memory. The new process merely executes the copy of `vi`'s code residing in main memory; the shared-code segment use count is incremented from one to two. When one of these processes finishes executing the code, the use count is decremented.

As long as the use count is greater than zero, the shared code remains in memory. When the last process finishes editing and terminates the `vi` program, however, the system decrements the use count to zero and releases `vi`'s shared code data structure and its associated physical memory.

Shared code was also designed to facilitate page alignment between main memory and disk, although pages are not guaranteed to be aligned.

Demand-Loaded Code (DEMAND_MAGIC)

Note For the Series 700/800, DEMAND_MAGIC code behaves identically to SHARE_MAGIC, despite a difference in the magic number.

Demand-loaded code encompasses some of the advantages of shared code and provides additional optimizations. (See comparison of shared and demand-loaded code in the next section.)

Like shared code, demand-loaded code addresses code and data separately. Demand-loaded code is also shared; only one copy of code need be in main memory for use by multiple processes.

When a user runs a demand-loadable program, pages of the program are read into memory only as required. The system also anticipates (from prior page usage) what subsequent pages might be required, and brings in additional pages.

Demand loading eliminates the need to allocate main memory to rarely accessed routines and code, such as error handling routines, which in some instances might account for a large percentage of a program's code.

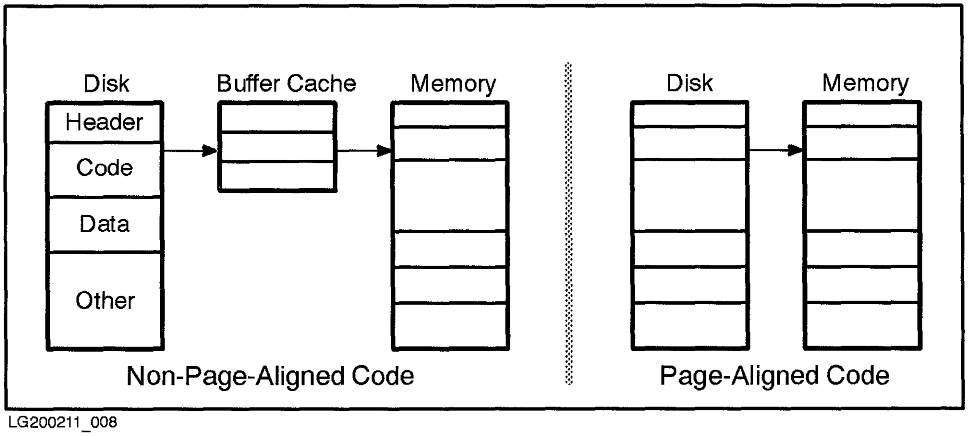


Figure 7-11.
Alignment of Page Boundaries Simplifies Transfer of Data and Code to Memory

Another demand-loaded code enhancement is guaranteed page alignment. This is illustrated by Figure 7-11. Guaranteed page alignment is based on a loading algorithm simpler for the system to implement. One-to-one mapping between paging device and main-memory pages allows for direct disk-to-memory transfer without an intermediary file-system buffer cache. Individual pages can also be copied faster.

7

When a executing process faults on a page, the process looks in the page cache (a data structure that describes every page of physical memory) for the page. If used in a recent process, the sought-after page is likely to be present. If the page is present, the kernel maps in its address for use by the process. If the page is not present, the kernel calculates the page's location from the inode (which was read into memory when the program began to execute) and maps in the page from disk.

Most executable code is now page aligned on 4KB page boundaries. Exceptions are older `a.out` files of Series 300/400 systems and executables that were compiled on Series 800 systems that were 2 KB-page aligned. When faulting in pages for executables whose code is not page aligned, pages must be copied in through the file-system buffer cache, a slower method than mapping through the page cache.

Benefits and Shortcomings of Shared and Demand-Loaded Code

If you work in an environment where applications are written in-house, you have some choice about how to link the object code for optimal performance. You might consider the following questions:

- How is the program linked by default?
- How do you expect the process's pages to be used?
  - At random?
  - Serially?
- Are your programs running on a system with ample or limited memory?
- Are there many or few users?

If an application program is running more slowly than expected, using *chatr*(1) or relinking to demand-loaded might improve its execution time.

Comparison of Shared and Demand-Loaded Code

The following points might help you determine which kind of executable to use:

- Most programs shipped as shared code by default.
  You can tell how the code is shipped by running the `file` command on the executable, for example:

  ```
  % file /bin/cat
  /bin/cat:        s800 shared executable
  ```
- Demand-loaded code gives flexibility; you can relink user code with *ld*(1) or mark programs demand-loaded with *chatr*(1).
- Both shared code and demand-loaded code reduce the amount of memory required for user code space when multiple process execute the same program.
- For both shared code and demand-loaded code, less memory required to run programs if only a subset of their pages are used, because pages are loaded only as needed.

Shared Code in an HP-UX Cluster (Series 300/400/700 only)

In HP-UX clusters, the benefits of shared code are reduced because each cluster node uses its own RAM. In other words, even though a piece of code may be marked as shared, if users on different cluster nodes invoke the program, each cluster node will still have its own copy of the code in its own memory. However, multiple users of a single cluster node would still share one copy of the code.

Shared Libraries

A **shared library** is a collection of commonly used subroutines located in one shared location in memory and which can be invoked dynamically at run-time by programs that need it.

The `libc`, `libm`, `libM` (the new POSIX-conformant math library), as well as some X and Starbase libraries are now provided in the `/lib` and `/usr/lib` directories as shared libraries. A set of shared library routines—`/usr/lib/libdld.sl`—also provides a user interface to the dynamic loader.

Shared libraries are distinguished from archived libraries by suffix; for example, the archive form of `libc` libraries are designated `libc.a`, whereas their shared-library counterparts are designated `libc.sl`.

7

A shared library reduces the amount of memory occupied by code during execution because only one copy of its routines exists in memory.

When you include a call to an **archived library**, on the other hand, the library code is copied to the executable file of the process at link time. Thus, each executable using a routine has a copy of that routine both on disk and in memory when running.

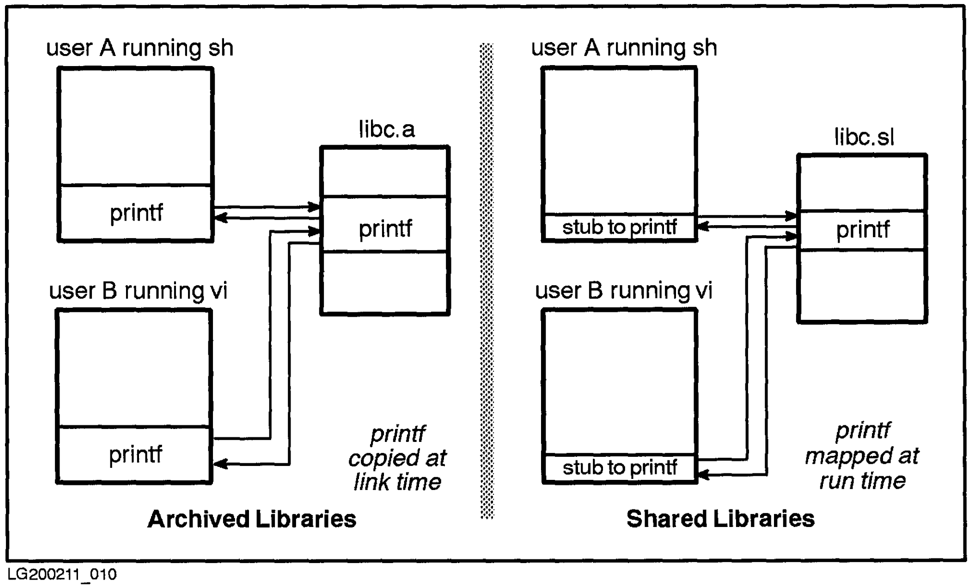


Figure 7-12. Shared Libraries Use Memory More Efficiently

Figure 7-12 illustrates the use of `printf` in `libc`. A program using archived libraries copies the `printf` library when linking `printf`. With shared libraries, `libc.sl` is loaded once, as a single entity. Only stubs to the `printf`'s offset within `libc.sl` are added to the address space of the user program. The calling program maps to `printf` at run-time.

Processes using shared libraries no longer require their own copy of the library. All processes calling the same library use the same image of that library, which is dynamically loaded and linked into executing process at run time. Even if several concurrent processes use the same shared library code, only one copy of the code exists in core memory. The memory image of a shared library is shared among all programs using that library.

How Shared Libraries are Dynamically Loaded

Shared libraries represent a change in the behavior of the compiled object-file code that gets executed before `main()`. That code (`crt0.o`) invokes the dynamic loader (`dld.sl`), which consults a table stored at the beginning of the process's code segment to determine what libraries to load. The executing program attaches all shared libraries specified on the linker command line. The library source is compiled as position-independent code. The object modules of this position-independent code are combined by the linker to form an object file given the extension `.sl`, for shared library.

When loaded into a system, shared libraries reside in main memory. Like demand-loaded code, only one physical copy of a shared library exists in memory; code and data are faulted in as needed. Library code is shared among all processes using that library. A private copy of library data is allocated for each process.

On attaching a shared library, the start-up code maps the shared library code and data into the process' address space, relocates any pointers in the shared library data that depend on the actual virtual address, allocates BSS, and binds all references into and out of the shared library by filling in linkage table entries in the program and the shared library.

To accelerate the program startup process, binding is normally deferred until the program actually calls the shared library routine. Each linkage table entry is initialized with a pointer to the dynamic loader. The first reference to each routine causes the dynamic loader to intercept the call and bind the reference to that routine. This deferred binding distributes the cost of symbol table lookup across the execution time of the program, and is especially useful if the program contains many references not likely to be executed.

The user can specify at link time that immediate binding be used. (On the Series 700/800, the `chatr` command has also been enhanced to allow you to specify binding mode without relinking.) Immediate binding causes all references to be bound at start-up time; thus, the cost of symbol table lookup is taken at startup rather than spread across the execution of the program. Immediate binding detects unresolved symbols at load time, rather than during the execution of the program. The default is deferred binding.

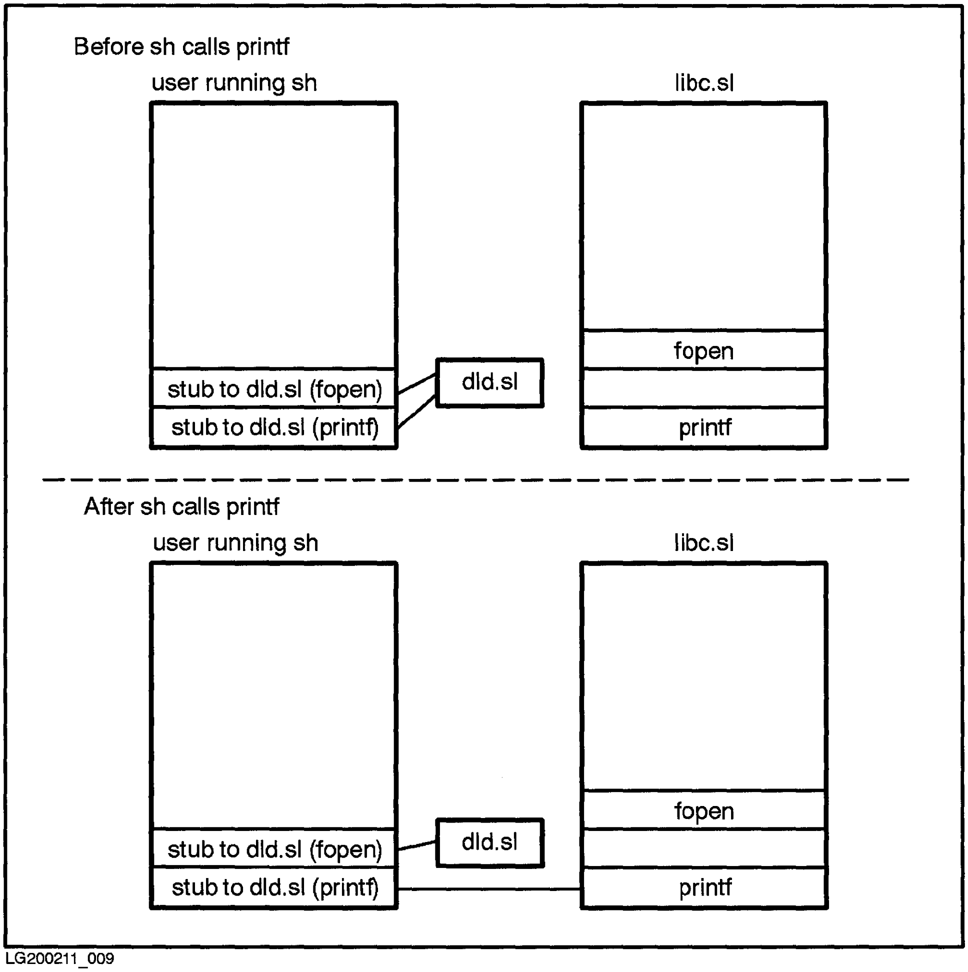


Figure 7-13. The Dynamic Loader at Work

Figure 7-13 shows how the dynamic loader (`dld.sl`) handles processes when a user invokes `sh`. A process is created with multiple stubs pointing to `dld.sl`. In this example (more specifically for Series 700/800), two stubs are shown—one stub for *fopen*(3S), other for *printf*(3S)—both of which are in the same library. (All the platforms—Series 300/400/700/800—use a linkage table as well; Series 700/800 use stubs also.)

When sh requires printf, the process first uses its stub to dld.sl, which does the work of establishing the pointer to printf and replacing its own stub with a stub to printf. Thereafter, when the process needs printf, a stub is in place to go right to printf, without having to point back to dld.sl.

Note that in the second half of the diagram, the stub intended for fopen is in place, although still pointing to dld.sl. However, it is ready to be accessed and replaced by a stub pointing to fopen.

Pointers also exist from the libc routines to dld.sl, but are not illustrated.

Shared Libraries vs. Archived Libraries

Occasionally, you may prefer to use an archived library when you link your application, in order to insulate your application from future modifications to the library. You can prevent a library change from adversely affecting your application by linking in the archived version of the library.

Usually, however, an application program benefits from updates to a library. Bug fixes and enhancements to a shared library result in automatic bug fixes and enhancements to a program that uses the library, and new features in a library should not affect an existing application. You can also use version numbers described in *Programming on HP-UX* to ensure that you execute a particular version when you do use shared libraries.

Another reason for using an archived library would be that the program will execute on another system that does not have the shared library. A program that uses a shared library cannot stand alone; the shared library must be present at run time. (An environment variable, SHLIB_PATH enables you to specify search path for shared libraries; see *Programming on HP-UX*.) By copying library code from an archive library into your program, you can prepare a program that is independent of any other system file.

Another difference between shared and archive libraries is performance. For object code to be shared, it must be position-independent; that is, the library cannot know where its code or data will reside while it is running, since different programs may place the library code and data at different addresses. (Text might vary the first time the library is loaded and stay fixed until the last reference is gone; data will be at a different address for each program.)

In most cases, the performance improvement resulting from using shared code and smaller executable files will outweigh the penalties of position-independent code. However, if your library will be used by only one running application at a time, it might perform better using an archive library.

Administering Shared Libraries

- Shared (not archived) libraries are the default implementation shipped. To use archived libraries when a shared library exists requires a special linker option.
- Consider carefully the path of shared libraries in the file-system tree. Unless the environment variable, `SHLIB_PATH` is used, the shared library must be found at run time at the same location (that is, at the same pathname) as it is found at link time, or the program will not run.
  - Be very careful where you put `/lib`, `/usr/lib`, `/usr/local/lib` and `/usr/contrib/lib`. Mistakes can be harrowing!

  For example, if `/lib` is not on the root volume, the system will try to use it before it is mounted. As a result, no `/etc/rc` commands (or their descendents) that depend on any libraries in `/lib` will be executable. Since `/etc/rc` uses many commands that depend on `/lib/dld.sl` and `/lib/libc.sl`, `/lib` must be in the root volume. (See *fs*(4) in the *HP-UX Reference* for more information on volumes and file-system structure.)
- You can NFS mount `/usr/local/lib`, but consider that all programs that use shared libraries from that directory will execute a little bit more slowly. (However, once the shared libraries are used, they remain in memory for a while; the NFS mount will then degrade performance only minimally.)
- For security, `/lib` and `/usr/lib` should *never* to writable by all. If `/usr/local` is writable by all, root should not be executing any programs that use shared libraries from `/usr/local/lib`. This precaution will prevent a potential security problem—actions of a malicious user, who could replace a library in that directory, thereby compromising any program using it.
- Missing or incorrect critical shared libraries can be detected during the boot procedure. If this occurs, the system administrator can recover the libraries using *recoversl*(1M) or via a backup tape.
- For a more thorough discussion of shared libraries, refer to *Programming on HP-UX*.

HP-UX Memory-Management Features

Besides providing fundamental support for virtual memory, HP-UX provides these important features:

- Shared memory for high-bandwidth interprocess communication (refer to *shmget*(2), *shmat*(2), and *shmctl*(2) in the *HP-UX Reference*).
- On Series 300/400/700 systems, device mapping for mapping physical addresses into virtual address space. This allows direct access to I/O devices (refer to *iomap*(7) and *graphics*(7) in the *HP-UX Reference*).
- Process locking for locking all or part of the user process space for real-time application needs (refer to *plock*(2) in the *HP-UX Reference*).
- On Series 300/400/700 systems, memory-mapped files allow applications to map file data into memory and perform I/O to files through direct loads and stores of memory instead of `read` and `write`. (Refer to *mmap*(2) in the *HP-UX Reference*.)

Swap Space Management

Swap space is an area on a high-speed storage device (almost always a disk drive), reserved for use by the virtual memory system for swapping and paging processes.

There are two types of swap space, device swap space and file-system swap space, both of which can be configured using SAM or the *swapon*(1M) command.

- **Device swap space** is located on a raw partition (or raw logical volume on a Series 800 system implementing LVM). Device swap is faster than file-system swap. On a system with traditional disk partitions, device swap space is fixed in size; on a system implementing LVM, the swap logical volume can be increased as needed. Typically, at least one swap device (primary swap) must be present on the system.
- **File-system swap space** is located on a mounted file system. File-system swap is slower, but varies in size with the system's swapping activity.

HP-UX swap-space management allows you to allocate swap as needed (that is, dynamically) while the system is running, without having to regenerate the kernel.

Total available swap on a system consists of all swap space available on all devices and file systems enabled as swap. The swapping subsystem reserves swap space at process creation time, but does not allocate swap space from the disk until swap time. Reserving swap at process creation protects the swapper from running out of swap space.

In a cluster (Series 300/400/700 only), multiple cluster nodes can share a single swap area. (Refer to Chapter 12 of *Managing Clusters of HP 9000 Computers: Sharing the HP-UX File System*.)

For sample swap-space configurations and procedures on setting them up, consult the *HP-UX System Administration Tasks* manual for your system.

A variation on traditional swapping, called **pseudo-swap reservation**, allows users to execute processes in memory without allocating physical swap. Pseudo-swap reservation is described at the end of this chapter.

Swap Parameters

HP-UX deals with swapping in terms of several parameters:

`swchunk`	The number of `DEV_BSIZE` blocks in a unit of swap space, by default, 2 MB on all systems. (For information on `DEV_BSIZE`, see Chapter 8, "HFS File System".)
`maxswapchunks`	Maximum number of swap chunks allowed on a system.
`min-swapchunks`	Number of chunks the system reserves for swapping at boot-up.
`swapmem_on`	Parameter allowing creation of more processes than you have swap space for. (See "Pseudo-Swap Reservation," at the end of this chapter.)

During system startup, the size and location of each swap device is displayed in 512-KB blocks:

`start =` *xxxxxx*	*indicates the swap space's starting disk block number*
`size =` *xxxxxx*	*indicates size of swap space*

Device Swap Space

Device swap space resides in its own reserved area—an entire disk, or a section or logical volume of a disk—and is not taken from file system space.

At least one swap device should be present on a system (unless you use pseudo-swap reservation). Device swap space is required during system startup.
7
You can also configure device swap space dynamically, without bringing the system into single-user mode. Unless you are using LVM (Series 800), device swap is fixed in size and location.

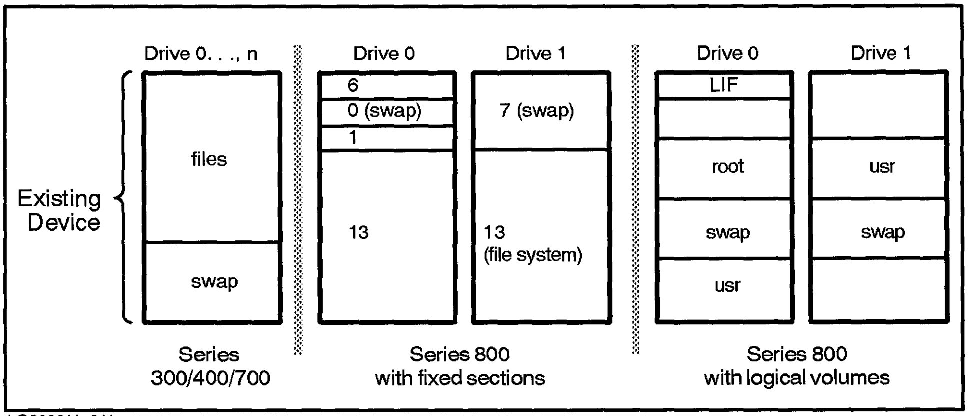
Using LVM, you cannot reduce the size of active device swap, because virtual memory cannot dynamically reconfigure the size of the swap device; thus, reducing the size of active device swap might cause a system panic. You can, however, increase the size of device swap by extending the swap logical volume as needed. However, the system will not recognize the change until you reboot.

One or more swap device can be configured into the system, by specifying the swap devices in the configuration file (`dfile` for Series 300/400/700 or `S800` for Series 800). You can also configure one swap device in the configuration file and dynamically configure more swap devices later, with `swapon`.

The first swap device listed in the configuration file is termed the **primary swap** device; that is, the swap space used first by the operating system. Primary swap defaults to the disk that contains the root file system.

By listing swap devices in `/etc/checklist`, you ensure that the swap devices are automatically enabled when the system is rebooted. The primary swap device might be listed in `/etc/checklist` as, for example, the device file `/dev/dsk/c0d0s15` (Series 800) where `c0` refers to the logical unit number 0, which is the first disk in the system, and `s15` is the section on that disk. Or, on a Series 800 system using LVM, the device might be listed as a logical volume, such as `/dev/vg00/rlvol2`.

The Series 300/400/700 and 800 systems handle disks differently, and this affects device swap space. On the Series 300/400/700, the entire disk is handled as a single section, and the *mkfs*(1M) command is used to apportion disk space for file system and swap. On the Series 800, disk space is divided either by fixed sections or by logical volumes. `swapon -a` is run as part of the `/etc/rc` script, which executes when you enter multi-user state. The `-a` instructs the system to read `/etc/checklist` for swap information and mount all swap sections.



LG200211_011

Figure 7-14. Series 300/400/700 and 800 Swap Space Compared

HP-UX allows you to configure swap on several disk drives, making it easy to expand the swap space. You can have multiple swap volumes per disk.

Having multiple swap devices on a system also increases throughput, by using the principle of "interleaving swap." Interleaving swap is achieved by assigning identical priority to multiple swap devices to minimize disk-head movement and enhance performance. (See Chapter 7, "Managing Swap Space" in *System Administration Tasks* manual.)

The required secondary storage for data, stack, and shared process segments is reserved from swap space during bootup. Therefore, swap space must be large enough to hold all segments of all existing processes. When it lacks sufficient swap space, the system either returns an error (such as `ENOMEM`) for some system calls, or kills the user process.

If you need more swap space, follow the procedures in the *System Administration Tasks* manual to do one of the following:

- Enable an additional swap device or file-system swap to your running system.
- (Series 300/400/700 only) Rebuild the file system on the existing device to reserve more swap space. To do this, you will also need to back up the existing file system, remake the file system using *mkfs*(1M), then restore the saved file system to the new configuration. For Series 700 only, run `mkboot` if this is the root disk. This must be done from another system.
- (Series 300/400/700 only) Enable swap area to be shared within a cluster. (See Chapter 12 of *Managing Clusters of HP 9000 Computers: Sharing the HP-UX File System* for procedures.)
- (Series 800 only) Rebuild the kernel using SAM or `uxgen` to change primary swap space. (You would do this if your needs have changed, and rather than continually allocating swap as needed, you decide to reorganize your disk space and allocate a different primary swap.)
- (Series 800 only) Create another logical volume or extend an existing swap logical volume. Note: to make use of the extended swap volume space, you must reboot the system.

File-System Swap

File-system swap, another form of secondary swap space, can also be configured dynamically. File-system swap space allows a process to use an existing file system if it needs more than the designated device swap space. The operating system swaps to space in the file system, in addition to device swap space. File-system swap is used only when device swap space is insufficient to meet demand-paging needs.

Whereas a swap device is limited in size to a specific section of a disk (or logical volume), file-system swap consumes a variable amount of space. This is an efficient use of disk space, because it enables the system to swap to unused portions of a file system as needed.

A process might need 20 MB of swap space for a short time. If only device swap is enabled, an entire 20 MB of swap space would have to be allocated permanently just to handle that brief work. With file-system swap, the paging system (that is, swapper" daemon) can fill these temporary needs from file-system space. The paging system and the file system share file system space. You can also limit file-system swap to a fixed size to prevent it from consuming too much space.

To optimize system performance, file-system swap space is allocated and de-allocated in `swchunk`-sized chunks. `swchunk` is a configurable operating system parameter; its default is 2048 KB (2 MB). Once a chunk of file system space is no longer being used by the paging system, it is released for file system use, unless it has been preallocated with `swapon`. (See *swapon*(1m) or *swapon*(2) in the *HP-UX Reference*.)

Guidelines for Adding Swap Space

Swap configured into the system is always available to a booted and running system. However, file-system and device swap can be allocated to a running system for short-term use, and is used by the system like any other configured swap device.

To make swap-space allocation automatic at boot time, you must include it in the `/etc/checklist` file. This ensures that the swap area is enabled when the system is rebooted.

If a swap area is not added to `/etc/checklist`, the system will no longer swap to that area after a reboot. To disable swap, you must edit `/etc/checklist` before rebooting. (These procedures can be performed using SAM.)

If you need more swap space but do not have any spare devices, we recommend that you add file-system swap space. System performance is nearly as good when using file-system swap space as device swap. If the system is using so much file-system swap space that performance degrades badly, you might want to increase device swap space.

If you have additional devices that can be used for swapping, enable these before file-system swap, because the performance is better. If using both device and file-system swap, give devices a higher priority (that is, a lower number in the *pri* argument when executing the `swapon` command). File system swap, given equal priority, is always used after device swap.

For instructions on adding swap space, see Chapter 7, "Managing Swap Space" in the *System Administration Tasks* manual and *swapon*(1M) in the *HP-UX Reference*.

Sample /etc/checklist Entry for Device Swap

To enable your swap device each time the system is rebooted, be sure to include an entry in `/etc/checklist`, as shown:

```
/dev/dsk/c1d0s1 /default swap pri=1
```

or when using LVM (Series 800 only):

```
/dev/vg00/lvol3 /default swap pri=1
```

Refer to the *checklist*(4) manual page of the *HP-UX Reference* for a thorough discussion of the fields used when adding device swap.

Sample /etc/checklist Entry for File-system Swap

To enable file-system swap when the system is rebooted, include an entry in `/etc/checklist`.

The following `/etc/checklist` entry enables swap on the file system containing the directory `/swap`. The size of a swap block is set by `swchunk`. It is currently set to 2048 bytes, in units of `dev_bsize`, which is 1024 bytes. (2048/size of swap block $\times$ 1024 bytes = 2MB, which is the allocation size of file-system swap-space chunks.) A minimum of ten file-system blocks are consumed immediately; a maximum of 4500 blocks can be used; 100 blocks are reserved for file system use; and the priority is 2:

```
default /swap swapfs min=10,lim=4500,res=100,pri=2
```

Refer to the *checklist*(4) manual page of the *HP-UX Reference* for a thorough discussion of the fields used when adding file-system swap.

Comparing Device and File-System Swap

Device swap is faster than file-system swap. This is because the system can write an entire request (64 KB on Series 300/400 or 256 KB on Series 700/800) to a device at once. File-system swap must be done in file-system blocks, which are typically 8 KB. As a result, the time it takes for one request in device swap might take longer using file-system swap. For example, a 2 MB chunk of file-system swap space is written to the disk in 256 requests in an 8 KB-block file system, or 32 requests in a 64 KB-block file system.

File-system swap makes more efficient use of file-system space, but it might degrade system performance somewhat, because its throughput is slower than device swap. This is because free file-system blocks may not always be contiguous; therefore, separate read/write requests must be made for each file-system block.

Swap Space Priorities

Priorities, ranging from zero to ten, can be set for all devices or file systems. The lower the number, the higher the priority. Thus, a device with a priority of zero is used for swapping before a device of priority one or higher.

Swapping rotates among both devices and file systems of equal priority. Given equal priority, however, devices are swapped to by the operating system before file systems, because devices make more efficient use of CPU time.

We recommend that you assign the same swapping priority to most swap devices, unless a device is significantly slower than the rest. Assigning equal priorities limits disk head movement, which improves swapping performance.

The swapon Command

The *swapon*(1M) command allows you to enable additional device or file system for paging and swapping. The swap space can be added while the system is running.

Sample Swap-Device Allocations

Example	Description
/etc/swapon -e /dev/dsk/3s0	Enables swapping to a block device on a Series 300/400/700 system using the space after the end of the file system (-e option) for swap and letting the priority default to 1.
/etc/swapon -p 0 /dev/dsk/c0d0s0	Enables swapping to a block device on a Series 800 system with logical number 0, CS/80 drive unit number 0, section 0, with the highest priority (zero).
/etc/swapon /dev/vg03/lvol4	Enables swapping on logical volume **lvol4** of volume group **vg03** on a Series 800 system using LVM. (See the section on Device Swap for limitations to changing the size of device swap. Also, see chapter 7, "Managing Logical Volumes" in the *System Administration Tasks* manual for use of LVM commands.)
/etc/swapon -p 2 -f /dev/dsk/c12d0s0	Forces swapping (**-f** option) to be enabled on Series 800 block device **/dev/dsk/c12d0s0**, even if a file system exists on the device. (Note that the **-f** option makes this a potentially *destructive* command, to be used with caution.) The device is assigned a swapping priority of 2.

7

Sample File-System Swap Allocations

Example	Description
`/etc/swapon /swap -m 256 -l 1024 -r 0 -p 3`	Enables file-system swap to a directory named **/swap**. Initially, 256 file-system blocks are consumed; no more than 1024 file-system blocks may be consumed. Also, 0 blocks are reserved for file-system use and this file system is assigned a swapping priority of 3.
`/etc/swapon /disk2 -m 0 -l 2048 -r 0 -p 10`	Enables file-system swap to a directory named **/disk2**. Initially, no (zero) file-system blocks are consumed; no more than 2048 file-system blocks maximum may be consumed. No file-system blocks are reserved for file-system use. The swapping priority is 10, meaning that **/disk2** is the least likely file system to be used for swap space.
`/etc/swapon /bigdisk -m 1024 -l 4096 -r 0 -p 0`	Enables file-system swap to a directory named **/bigdisk**. Initially, 1024 file system blocks are consumed, no more than a maximum of 4096 file-system blocks may be consumed. No file-system blocks are reserved. **/bigdisk** is assigned a swapping priority of 0, meaning it will be used most often for swap space.
`/etc/swapon /dyndisk -m 0 -l 0 -r 2048 -p 0`	Enables file-system swap to a directory named **/dyndisk**. Although no file-system space is allocated initially, no limit is set on the amount of space that can be used. 2048 blocks are reserved for file system use. **/dyndisk** has a swapping priority of 0—the highest.

Evaluating Swap-Space Needs

As a system administrator, you need to monitor your system's swap space regularly, because swap space usage varies with system load. You want to understand your system's swap requirements when demand is heaviest.

Swap space must be large enough to hold the sum of all shared memory, shared libraries, stack, and data for the largest executable process.

The `size` command is a useful tool for acquiring information on process size:

```
% size /usr/bin/vi
243332 + 222992 + 147620 = 423944
```

`size` returns the code (text), data, and `bss` (data uninitialized at the beginning of process execution) for a program. Since no swap space is reserved for code (the first figure), you would combine only the second and third sizes to estimate the size a program occupies in the swap area. Multiply the figure by the number of users executing the program at the system's busiest time, since each user is allocated data and `bss`. Repeat this calculation for each program executed when the system is at its busiest.

Shared memory must also be included in your swap-space needs assessment. To display the amount of shared memory your system is using at any given moment, use the `ipcs` command:

```
%  ipcs -b
IPC status from /dev/kmem as of Fri May 29 14:53:42 1992
T      ID      KEY          MODE        OWNER     GROUP QBYTES
Message Queues:
q       0 0x3c341834 -Rrw--w--w-     root      root  16384
q       1 0x3e341834 --rw-r--r--     root      root    264
T      ID      KEY          MODE        OWNER     GROUP  SEGSZ
Shared Memory:
m       0 0x41341837 --rw-rw-rw-     root      root    512
m       1 0x4134184f --rw-rw-rw-     root      root   7452
m       2 0x411800ac --rw-rw-rw-     root      root   8192
T      ID      KEY          MODE        OWNER     GROUP NSEMS
Semaphores:
s       0 0x4134184f --ra-ra-ra-     root      root      2
```

In this example, 512, 7452, and 8192 are shared memory segment sizes. You would add them to the data and `bss` sizes. To account conservatively for page boundaries, round up your estimate.

Because the swap allocation unit, **swchunk**, is by default 2MB, your figure should be a multiple of 2MB. If you do not use a multiple of **swchunk** some swap space is wasted.

The value of **swchunk** is derived as follows: A chunk is an integer with a default value of 2048. Multiplying that integer by the default block size of 1024 bytes found in **/etc/disktab**, you derive the swap chunk size of 2 MB.

The figure of 2 MB is then multiplied by the operating-system parameter, **maxswapchunks**, to derive the limit of swap space allowed on your system. **maxswapchunks** is by default 256 (2^8), but can be set to any value up to 2^{14}. The value specified does not have to be a power of 2.

One more parameter should be considered when determining swap-space requirements. While overlaying the old process image with the new process image, the `exec` system call uses an area of swap space to temporarily hold arguments and environment variables. The size of this area is determined by the configurable system parameter **argdevnblks**, whose default size is 256 KB.

If you need to change the amount of swap space to accomodate an application, refer to the application's manual to see if it describes swap space requirements.

For details on setting and changing device and file-system swap space, see the *System Administration Tasks* manual.

7

Swapping in an HP-UX Cluster (Series 300/400/700 only)

Swapping on a cluster server is identical to swapping on a standalone system: the server must swap to a disk that is physically connected to it. A cluster client can do any one of the following:

- Swap to the server's swap area.
- Swap to a swap area on the client's own local disk.
- Swap to a swap area on another client's local disk.

To set up local swap in a clustered environment, see *Managing Clusters of HP9000 Computers: Sharing the HP-UX File System*.

Pseudo-Swap Reservation (Series 800 only)

HP-UX (Series 800 only) has a new operating-system parameter to provide the capability of using system memory for swap space. By default, `swapmem_on` is set to 1, enabling **pseudo-swap reservation**.

Typically, some physical memory (such as a disk) must be configured for swap space. When the system executes a process, swap space is usually reserved for the entire process, in case the process must be swapped out. According to this model, to run one gigabyte of processes, the system would have to have one gigabyte of configured swap space. Although this protects the system from running out of swap space, disk space reserved for swap is under-utilized if minimal or no swapping occurs.

To avoid such waste of resources, HP-UX swap space is now configured to access device swap space, file-system swap space, *plus* up to three-quarters of system memory capacity. This means that system memory serves two functions: as process-execution space and as swap space.

System memory used for swap space is called **pseudo-swap** space. By using pseudo-swap space, a one-gigabyte memory system with one-gigabyte of swap can run up to 1.75 GB of processes. As before, if any process attempts to grow or be created beyond this extended threshold, the process will fail.

Pseudo-swap space is set to a maximum of three-quarters of system memory because the system starts paging once three-quarters of system available memory has been used (see "Maintaining Page Availability—vhand and swapper Daemons").

For factory-floor systems (such as controllers), which perform best when the entire application is resident in memory, pseudo-swap space can be used to enhance performance: you can either lock the application in memory or make sure the total number of processes created does not exceed three-quarters of system memory.

Note, however, that when the number of processes created approaches capacity, the system might exhibit thrashing and a decrease in system response time. If necessary, you can disable pseudo-swap space by setting the tunable parameter `swapmem_on` in `/etc/master` to zero.

8

HFS File System

HP-UX uses a file system called the High Performance File System (HFS), which is also known as the McKusick (or BSD) file system. *A Beginner's Guide to HP-UX* discusses the file system at the user level; this chapter describes the structure of the file system and its relationship to the disks on which file systems reside.

The following additional resources are useful in developing an in-depth understanding of the HFS file system and how to administer it:

- *System Administration Tasks* manual, for creating and managing file systems and disk space.
- *Solving HP-UX Problems* for repairing file systems and dealing with problems that arise in managing file systems and disk space.
- section (4) of the *HP-UX Reference*, for specifications of HFS file-system formats.

To work effectively with file systems, you must understand their interrelationship with physical disks. Every file of the HFS file system is stored on a formatted mass storage medium, a disk. The disk is known to HP-UX by specifying the path name to the disk's device file. Device drivers in the operating system enable communication to the disk. Each architecture supports a different set of disks, based on the device drivers written for that architecture and disk. To access files in a file system, you mount the file system on a disk, by associating the path name of its mount point to the disk's device file. Once mounted, the file system is accessible to the operating system and users.

This chapter discusses file-system creation, storage, modification, and protection.

Understanding File-System Creation

As a system administrator, much of what you do concerns file systems. System files, application files, and user files are typically organized as file systems. Also, although disks are the storage devices that hold data, the data must reside in a file system to be available to the operating system. Thus, if you run short of space, you can install a new disk and create a file system on it to hold additional data.

Conceptually, the creating a file system involves:

- making the physical environment (the disk device) available to the file system.
- creating the software entity (the file system) itself.
- establishing (by mounting) the "connective threads" between the physical and software elements.

HP-UX uses the term "file system" to mean several things: A file system is the HP-UX file-system (often several file systems mounted together) directory tree, starting from root. File system is also a body of structures that exist on each file-system device that enables you to keep data contiguous with the existing data hierarchy. This second meaning of file system is the subject of this chapter. This section summarizes the numerous aspects of file system creation, to explain how a file system is connected to HP-UX as a whole.

Note All procedures for creating and maintaining file systems are found in *System Administration Tasks* manual, Chapter 5, "Managing the File System."

There are many reasons why you might add a new file system, including:

8

- You anticipate that your file system will soon exceed current maximum capacity.
- Your current file system has already reached maximum capacity.
- You wish to separate portions of a file system physically, to restrict growth of files on a portion of the file system or to increase concurrent access for better performance.

To create a file system, you can use a sequence of HP-UX commands, or you can invoke the SAM utility and perform the task interactively. In either case, adding a file system involves:

- Installing the necessary device files for the new device (done if disk is newly connected)
- Preparing the storage medium (the disk device) for the file system (if disk is newly connected)
- Creating the file system itself.
- Mounting the file system to make it available for system use.
- Adding the file system to `/etc/checklist` for automatic mounting.

If you are creating your new file system on a new disk drive, you first connect the physical device to the system, referring to the device's installation manual. Use a hard disk to hold an HP-UX file system. The capacity of flexible disks, cartridge and reel tape drives is too limited, slow, and subject to deterioration from such constant use. Rewritable magneto-optical disks are slower than hard disks, but substantially faster than flexible disks or tape, and are typically used to back up a file system. If necessary, magneto-optical disks can be used to hold an auxiliary file system.

Figure 8-1 shows the physical and logical orientation of a file system to your overall computing environment.

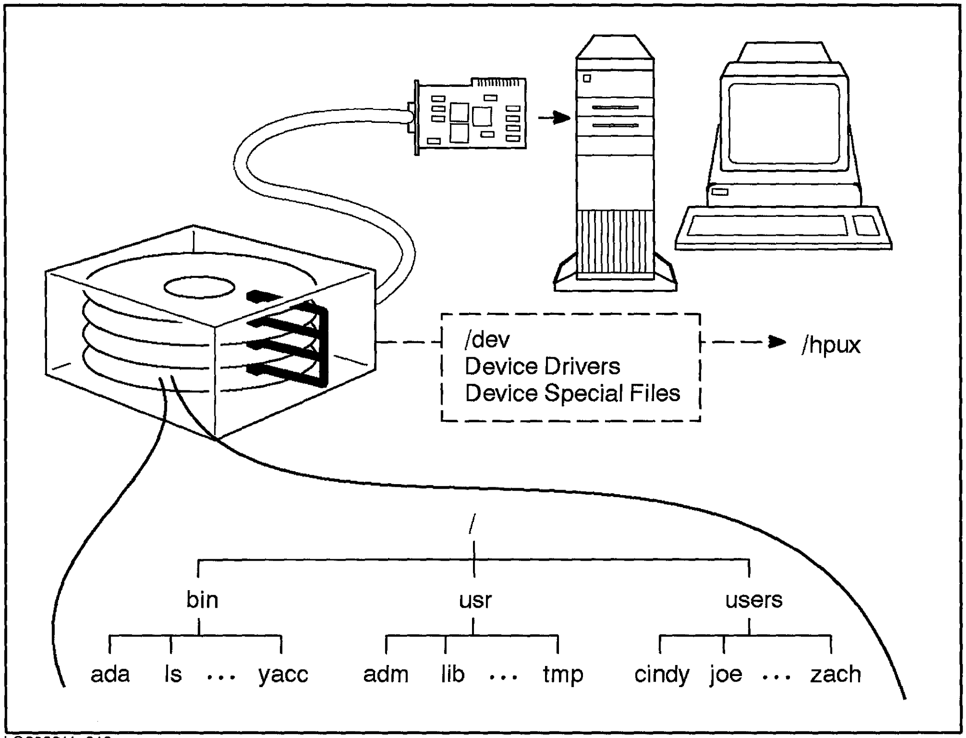


Figure 8-1. File System, Viewed Physically and Logically

Each type of disk is accessed *physically* via a compatible interface card that connects the disk to the computer's bus architecture. Hard disk drives might use any of the following interfaces—standard or high-speed HP-IB, fiber link (HP-FL), or small computer systems interface (SCSI). The protocol for each interface is encoded in a specific device driver. (See Chapter 9, "System Configuration," for information on device drivers, device files, and configuration concepts.)

8

The operating system accesses physical devices *logically* through both the device driver and device special files.

- You can see the device drivers used by your system by reading the `dfile` (for Series 300, 400, or 700 systems) or the `S800` file (for Series 800 systems), or by running the *lsdev*(1M) command.
- You can see device special files for disks by listing the `/dev/dsk` (for block special files) and `/dev/rdsk` (for character special files) directories.

You create device files using the *mknod*(1M) command. For Series 800 systems, you can also create device files using the *mksf*(1M) or *insf*(1M) commands. (On the Series 800, device files for all devices the system finds are created by `insf`, which is run as the `-i` option of `ioinit` by the `/etc/inittab` file at boot-up time. Thus, you should never create device files manually for a new disk, unless it was not connected or powered on at boot time.)

Both character and block device special files are required for devices that hold file systems. Thus, for Series 300, 400, and 700 systems, you need a character and block device special file for the entire disk drive. For Series 800 systems, you need a character and block device special file for each section of the disk drive in use. Or, on Series 800 systems, if you are apportioning disk space using the Logical Volume Manager (LVM), you need a character and block device special file for each logical volume. (LVM is the subject of chapters in both this manual and the *System Administration Tasks* manual.)

The device special files are used when performing system administration tasks involving the file system. For example,

- The *mediainit*(1M) command requires a transparent special file to reformat a disk or tape for a file system. Use `mediainit` if you suspect the media is corrupted or worn. To use `mediainit`, you must create the device files using the `-t` of *mksf*(1M).
- The *mount*(1M) requires block device files to mount and unmount (`umount`) the file system.
- The *newfs*(1M) command requires a character special file to create a file system.

In Figure 8-2, you can see the role of `newfs` in preparing the disk to accept a file system.

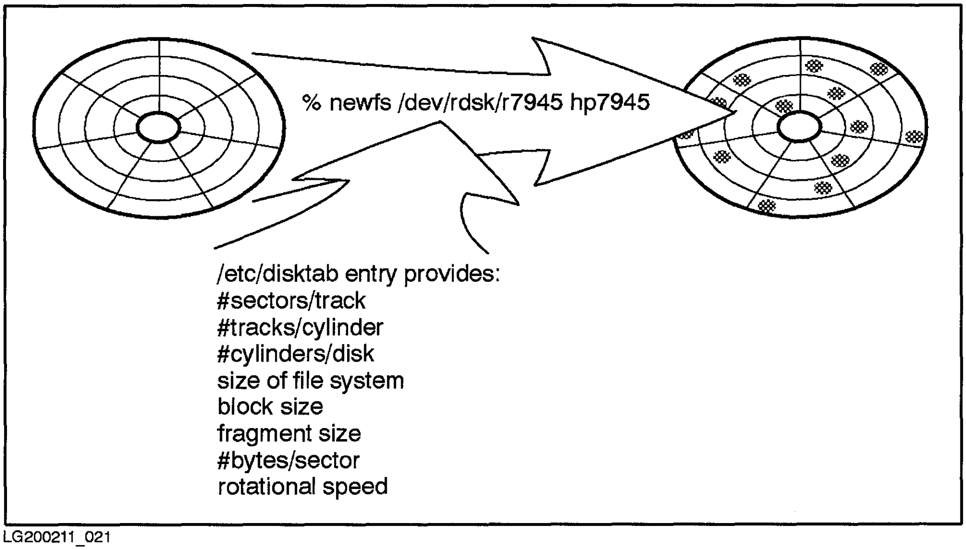


Figure 8-2. newfs Builds the Disk Infrastructure for a File System

HP-UX cannot use media to store files until you place a file system on it. You can create a new file system using SAM, *mkfs*(1M), or *newfs*(1M). Of the two manual commands, `newfs` is easier to use. When you create a file system, you create an environment to contain files, much like building a "file cabinet" for paper files. When first built, the file cabinet is empty. Then you add files.

To create a file system, you specify disk type to the `newfs` command, which then locates it in the `/etc/disktab` file of disk characteristics and extracts key values, including block and fragment size, number of bytes per inode, percentage of reserved free space, and rotational delay. The `newfs` command uses this information to create the infrastructure on the media necessary to support the file system's data structures.

Procedures for building a file system are documented in *System Administration Tasks* manual, Chapter 5, section entitled, "Creating File Systems Using the `newfs` Command."

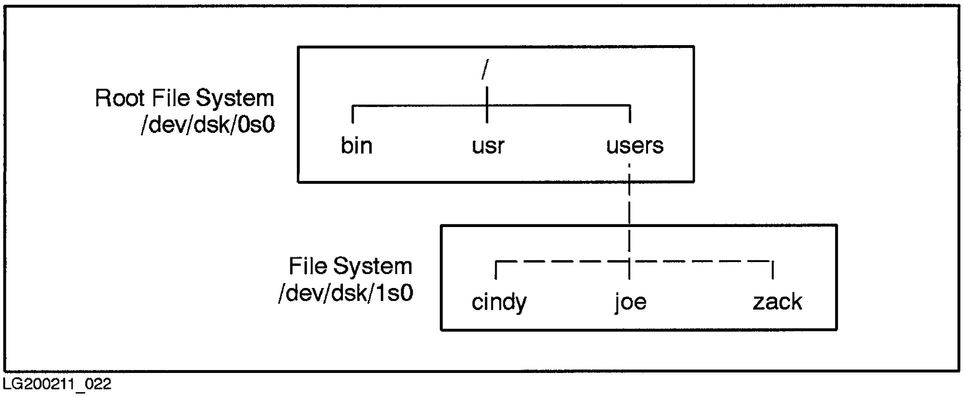


Figure 8-3. File System /dev/dsk/1s0 Mounted to Root File System at /users

After creating a file system, the file system has to be mounted (attached) to the HP-UX file hierarchy, using the *mount*(1M) command. This incorporates the file system into the existing file system's overall hierarchy. You do this by logically associating the root directory of the new file system with a mount point, a directory on the existing file system. Once a file system is mounted, the mount points are seamless. You can access the new mounted disk space as a contiguous part of the entire HP-UX file-system hierarchy. Figure 8-3 shows a mounted file system.

To mount a file system:

- make a mount point directory (using the `mkdir` command) for the file system.
- mount the newly created file system to the mount point (using the `mount` command).

If need be, an existing file system can be moved to a different location on the HP-UX file hierarchy by unmounting (detaching) it from its current location using the `-u` option (or `umount` command) of *mount*(1M) and remounting the file system. A file system cannot be unmounted if any files are open or if any user's current working directory is in that file system. You can use the *fuser*(1M) command to identify which processes are using a file system or file structure, and if necessary, terminate them. The `shutdown` command unmounts all mounted file systems before bringing a system down, so that the file systems are not corrupted.

You cannot unmount the root file system or any file system that has dynamic swap enabled. Likewise, be sure that the `/etc`, `/dev`, and `/bin` directories are mounted directly on the root file system, so that they cannot be inadvertently unmounted.

For mounting, you refer to the file system by its section's device file name (or with LVM, its logical volume) and its mount point directory. For unmounting, you refer to the file system by either the device file name or mount point, because unmounting breaks the link between the two.

As a system administrator, you maintain the `/etc/checklist` file as a record of mountable file systems and swap space. The `/etc/checklist` file is read:

- by `/etc/rc`, to mount all listed file systems when the system is booted up.
- by *fsck*(1M), to determine the order for conducting file-system checks.
- by *shutdown*(1M), to unmount all file systems before halting the system.
- by library calls such as *getfsent*(3X) and *getmntent*(3X), which enable programs to make use of file system information.

Disk Layout

The disk layout is the geometry applied to a physical disk. For each platform, the `/etc/disktab` file lists the disk geometry of each disk model available.

Typically, a disk is divided into areas that can accommodate file systems or raw I/O, dump, and swap. A disk from which the system can be booted is called a root disk, is organized somewhat differently from other disks, and is discussed later in this chapter. Non-root disks typically contain a single swap area, file systems, or a combination of both.

The following sections show the layouts of HP-UX disks for each architecture. Note, only Series 800 disks support disk sectioning and logical volume management.

Series 300/400 Disk Layout

On Series 300/400, a root disk consists of three areas, each discussed later in this chapter:

- An 8-KB boot area, which contains a small file system in Logical Interchange Format (LIF) and the secondary loader, which loads in the HP-UX kernel, `/hp-ux`.
- A file system, consisting of a primary superblock and one or more cylinder groups. (Superblocks and cylinder groups are discussed in a later section entitled "File system Layout.")
- An area reserved for swap.

Most Series 300/400 disks cannot be not partitioned; each disk has at most one area for swap, file system, and boot. However, for Series 300/400 disks that *do* support "hard" partitions, such as the HP 9133H, each partition can be addressed separately; the volume field in the minor address indicates the partition. Treat these partitions as though they are several separate disks, each conforming to the layout shown in Table 8-1.

Table 8-1. Series 300/400 Root Disk Layout

Area	Data Structure	Size
Boot area	LIF file system	8 KB
File system and Dynamic Swap	Superblocks[1]	8 KB
	Cylinder group 1	varies
	Cylinder group 2	
	⋮	
	Cylinder group *n*	
Reserved swap area	Swap tables[2]	0 or more blocks

1 primary and redundant.

2 structures defined in `/usr/include/sys/swap.h`

8

Series 700 Disk Layout

Except for the (optional) boot area, the Series 700 disk is laid out much like the Series 300/400, as shown in Table 8-2.

- The LIF directory, the first 8 KB of the Series 700, contains pointers to the file system and to each boot program in the boot area.
- File system and swap areas resemble those of the Series 300/400.
- In the absence of a boot area, the swap area occupies the remaining space.
- The Series 700 boot program occupies the last 2 MB of the root disk layout.

Table 8-2. Series 700 Disk Layout

Area	Data Structure	Size
Boot pointers	LIF directory	8 KB
File system and Dynamic Swap	Superblocks[1]	8 KB
	Cylinder group 1	varies
	Cylinder group 2	
	⋮	
	Cylinder group *n*	
Swap	Swap tables[2]	0 or more blocks
Boot area[3]	LIF file system	2 MB

1 primary and redundant.

2 structures defined in `/usr/include/sys/swap.h`

3 optional.

A Series 700 feature, Software Disk Striping (SDS), groups disks into arrays; data on these disks can then be managed as if occupying a single disk. For full information, see *sdsadmin*(1M) in the *HP-UX Reference Manual* and the manual, *Improving Performance with Software Disk Striping*.

Series 800 Disk Layout

Series 800 disks differ from those of Series 300/400/700 because they can be apportioned into sections (partitions) to hold multiple file systems.

Using the logical volume manager (LVM) on the Series 800, you can also configure your disks with logical volumes, which are similar to disk sections, but are more flexible. (You can span disks with logical volumes, and extend or reduce their size as needs change.) LVM is the subject of a separate chapter in this manual, as well as in the *System Administration Tasks* manual.

Because Series 800 uses disk sections, both the boot area and primary swap occupy their own disk sections.

Disk space can be partitioned on the Series 800 in a variety of ways, as discussed in the next section.

Understanding Disk Partitioning (Series 800 only)

Disk partitioning is a Series 800 feature that allows you to divide a disk into **sections**, which can be addressed like separate disk drives. Disk sections allow the system administrator to manage disk space more precisely than on a system without disk sections.

A section can used for:

- Boot area
- File system
- Swap area
- Raw I/O

The layout of each section is nearly identical to the same areas on the Series 300/400/700. However, on the Series 800, the boot and swap areas reside in their own sections instead of residing in the same section as the file system.

As of HP-UX 9.0, the Series 800 can be partitioned using two different methods:

- HP-UX disk sections, the traditional disk partitioning method.
- Logical volumes, managed by the LVM.

HP-UX Disk Sections

Series 800 disks are partitioned into sixteen possible section choices. The size and location of each section is dependent on the disk model. Section sizes and locations are defined in the `/etc/disktab` file (described in further detail in a coming section) for most supported disk models. Figure 8-4 shows the choices of hard-coded sections available for Series 800 disks. (Note that disks smaller than 350 MB have a slightly different layout. See `/etc/disktab` on your system for layout.)

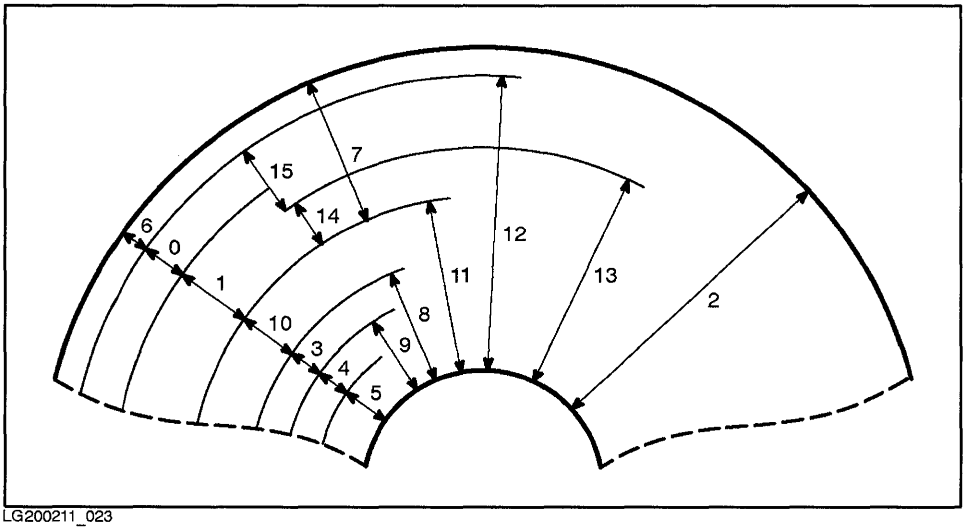


Figure 8-4. Disk Sections and Relative Sizes

8

When you create a new file system using the **mkfs** or **newfs** commands, or the SAM program, you declare what section the file system is to be mounted on. (For procedures on creating a file system, see the *System Administration Tasks* manual.) You must be careful not to use overlapping disk sections (as shown in Figure 8-4 and at the beginning of the `/etc/disktab` file).

For example, if you create a file system on section 11 on the 7935 disk, you *cannot* allocate sections 2, 3, 4, 5, 8, 9, 10, 12, and 13 for other file systems, because they would conflict with section 11. You *can* put file systems in sections 1, 0, and 6, or 6, 14, and 15.

Figure 8-5 shows two valid ways to set up disk sections. Compare this figure with Figure 8-4 to understand how to make the most of disk sections.

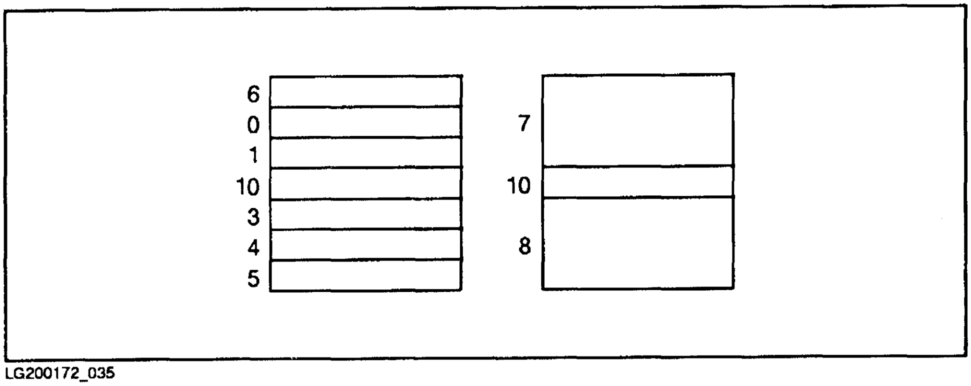


Figure 8-5. Examples of Valid Sectioning Schemes

Logical Volumes

The Logical Volume Manager (LVM), available on Series 800 systems, enables you to partition disks in a more flexible manner than with traditional disk sections. Using LVM, you combine one or more disks (physical volumes) into a volume group, which can then be subdivided into logical volumes.

Logical volumes resemble disk sections, but with some important differences:

- You can define the size of a logical volume according to need.
- You can extend or reduce the size of logical volume as needs change.
- Logical volumes can span disks. This enables you to create very large logical volumes, or use small portions of disk space more efficiently.
- You can mirror logical volumes, using an optional product, LVM Mirroring.

Logical Volume Manager (LVM) is the subject of a separate chapter in this manual and in *System Administration Tasks* manual.

File System Size

HP-UX supports file systems up to 4 GB. Note, however, that the size limit for individual files is still 2 GB. Applications may also not use raw access to disk sections larger than 2 GB.

For very large disks (such as HP C2254B), the boot partition must lie within 2 GB of the beginning of the disk.

Protocols do not permit NFS-mounting file systems larger than 2 GB.

Disk and File System Tools

When working with file systems, you often have to understand how much disk space you have and how large your file systems are. Two HP-UX tools—`/etc/disktab` and `/etc/diskinfo`—can help you determine available disk space. To view how large a file system is that you want to mount, you can use `bdf`, `df` or `du`. Each of these tools is discussed in the following sections.

Disk Geometry Database—/etc/disktab

The `/etc/disktab` file exists on all systems as a database and informational file about disks. It provides you with both reference information about the many HP disks supported on a given computer system and tutorial information about disk geometry.

Because it is a database (and thus written for the system, rather than to be read by people), the information in `/etc/disktab` appears in terse form, as follows:

`ty`	Type of disk.
`ns`	Number of 1K sectors per track.
`nt`	Number of tracks per cylinder.
`nc`	Total number of cylinders per disk.
`s0`	Size of file system (Series 300/400/700) or section (Series 800) in 1K blocks. On Series 800, `s0` designates section 0; subsequent sections are listed `s1` through `s15`.

`b0`	Series 300/400/700: Block size in bytes. Series 800: Block size of section 0 in bytes. Subsequent block sizes are listed `b1` through `b15`, corresponding to their respective section. Default block size for all systems is 8K.
`f0`	Series 300/400/700: Fragment sizes in bytes (1K, 2K, or 4K are supported). Series 800: Fragment size of section 0 in bytes. Subsequent fragment sizes are listed `f1` through `f15`, corresponding to their respective section. Default fragment size for all systems is 1K.
`se`	Number of bytes per physical sector.
`rm`	Rotational speed of disk platters by revolutions per minute.

Not all abbreviations are used on all systems.

The contents of `/etc/disktab` are used by the `newfs` command. On the Series 300/400/700, `/etc/disktab` provides multiple entries for any given disk, to enable you to specify to `newfs` whether you want portions of a disk used for swap and boot.

If you are partitioning your disks using traditional disk sections, the Series 800 `/etc/disktab` file is useful for determining valid disk sections because it shows Figure 8-4, from which you can ensure that you neither overlap disk sections or leave gaps. `/etc/disktab`'s listing of each disk with the size of each section enables you to calculate available disk space.

If you are using the LVM, you have less cause to refer to `/etc/disktab`, although you might refer to it when you want to use non-default settings for file-system specification (for example, to change the fragment size, customize the various file-system sizes). Before adding a physical volume (disk) to a volume group, you might consult `/etc/disktab` to get an idea of the disk size. Be forewarned, however, that once you add the physical volume, the LVM uses a portion of the disk (as much as 1 MB) for its data structures, so the size provided by reading section 2 of `/etc/disktab` for a given disk is larger than the actual size you have available.

For full specifications, see *disktab*(4) in the *HP-UX Reference*.

Disk Characteristics Command—/etc/diskinfo

The *diskinfo*(1M) command displays characteristics of a disk device, when given the device's character special file. `/etc/diskinfo` is particularly useful when setting up or managing logical volumes.

When used without options, `/etc/diskinfo` produces terse output:

```
% /etc/diskinfo /dev/rdsk/c0d0s2

CS80 describe of /dev/rdsk/c0d0s2:
         product id: 7937
               type: fixed disc
               size: 571444224 bytes
   bytes per sector: 256
```

With the `-b` option, `/etc/diskinfo` returns the size of the disk in 1024-byte sectors.

```
% /etc/diskinfo -b /dev/rdsk/c0d0s2

558051
```

The verbose (`-v`) option of `/etc/diskinfo` displays different information, depending on type of disk:

8

- vendor and product ID (SCSI devices)
- device name (CS/80 and SCSI)
- number of bytes/sector (CS/80 and SCSI)
- geometry, interleave, and timing information (CS/80)
- size in bytes and logical blocks, revision level, SCSI conformance level (SCSI)

```
% /etc/diskinfo -v /dev/rdsk/c0d0s2

CS80 describe of /dev/rdsk/c0d0s2:
controller:  0x8001  installed unit word
               1250  max instantaneous xfr rate (Kbytes/sec)
                  0  CS/80 integrated single unit controller
unit:             0  generic device type (fixed disc)
             079370  device number (6 bcd digits)
                256  # of bytes per block
                128  # of blocks buffered
                  0  recommended burst size
                179  blocktime (microseconds)
               1000  continuous average transfer rate (Kbytes/sec)
                 80  optimal retry time (centiseconds)
                 84  access time parameter (centiseconds)
                  1  maximum interleave factor
               0x01  fixed volume byte
               0x00  removable volume byte
volume:        1395  maximum cylinder address
                 12  maximum head address
                122  maximum sector address
            2232203  maximum single-vector address
                  1  current interleave factor
```

Free Disk Blocks Command—bdf

The `bdf` command (Berkeley's variation of `df`) reports the number of free disk blocks available on a file system. If no file system is given as an argument, `bdf` reports on all file systems. Several options are available:

`-b`	Displays information about file system swapping.
`-i`	Displays used and free inodes.
`-l`	Local. Displays HFS and CDFS file systems mounted on a client. It does not display NFS-mounted file systems.
`-L`	Displays file systems that can be unmounted from a client, including NFS-mounted file systems.
`-t` *type*	Displays only information on mounted file systems of a given type.

Here is sample output of `bdf`:

```
% bdf

Filesystem          kbytes    used   avail capacity Mounted on
/dev/dsk/c0d0s13    484960  242945  193519    56%   /
/dev/dsk/c5d0s8     237810   48481  165548    23%   /graphics
/dev/dsk/c4d0s2     277954  130729  133327    50%   /users
/dev/dsk/c3d0s10    121663   21878   87618    20%   /tmp
/dev/dsk/c2d0s2     277954  195693   54465    78%   /projects
/dev/dsk/c1d0s11    461664  320288   95209    77%   /usr
/dev/dsk/c3d0s7      70979    2994   60887     5%   /usr/spool
/dev/dsk/c1d0s7      71599   44562   23457    66%   /usr/contrib
```

`bdf` reports its output in 1024-byte blocks.
`df` reports its output in 512-byte blocks.

8

Disk Usage Command—du

The du command reports disk usage in 512-byte blocks for all files or directories specified; if none is specified, du reports on the current directory. Its report traverses the file tree recursively.

Here is sample output using du on one of the file systems listed in the previous example:

```
% du /usr/contrib
16      /usr/contrib/lost+found
448     /usr/contrib/bin+/HP-PA/respond+
2832    /usr/contrib/bin+/HP-PA
3384    /usr/contrib/bin+/HP-MC68020
6218    /usr/contrib/bin+
2       /usr/contrib/etc
2       /usr/contrib/games
2       /usr/contrib/include
 .
 .
 .
486     /usr/contrib/system
8       /usr/contrib/xseethru
89124   /usr/contrib
```

The final number reported is the total of *512*-byte blocks for the `/usr/contrib` file system, and therefore the number is twice as large as that reported by `bdf` in *1024*-byte blocks.

Note If it encounters a protected directory (that is, one whose file permissions are set to prevent access), du cannot report the number of blocks contained in that directory or its subdirectories.

Boot Area

The **boot area** is the portion of the disk that holds the code used to bring the system into an operational state. The boot code initializes and tests the hardware, then loads into memory a secondary loader. The secondary loader is the program that loads `/hp-ux` (the operating system) into memory to enable you to use your system. (For detailed information about boot code, see Chapter 2, "System Startup.")

The boot area is reserved on the mass storage medium (usually a disk) during the installation process. Information in the boot area is used only if the disk is used for booting (boot disk), but the space can be reserved on all disks.

On Series 300/400/700, the boot area precedes the file system on the disk.

On Series 800 systems using traditional disk sections, the boot area must reside in its own disk section (section 6) distinct from the file system and swap area sections.

Typically, section 6 is used since it is one of the smallest sections, yet large enough to contain the boot area. On Series 800 systems using the Logical Volume Manager (LVM), section 6 might not be large enough, so section 2 is always used.

Using LVM, the boot data is contained in a Boot Data Reserved Area, which is created using the `pvcreate -B` command. (For detailed information on LVM and the Boot Data Reserved Area, see Chapter 9, "Logical Volume Manager (LVM)," of this manual. For procedures on using LVM, see Chapter 7 of *System Administration Tasks* manual.)

Although the disk layouts for the various HP-UX platforms differ, all systems (Series 300/400/700/800) use a small file system for the initial system booting, written in Logical Interchange Format (LIF). (LIF is described in *lif*(4) in
8 the *HP-UX Reference*. The manual page also contains pointers to the LIF utilities.)

Series 300/400 Boot Area Implementation

On the Series 300/400 only, the boot area contains a volume header, volume directory information, and a small secondary loader used when the system is loaded. This area is reserved exclusively for use by the boot ROM. Table 8-3 shows the layout of this area, and the size of its components.

Table 8-3. Series 300/400 Boot Area Layout

Subarea	Size
LIF volume header	256 bytes
(unused)	256 bytes
Volume directory information	256 bytes
Bootstrap program	7.25 KB

Unless you create your file system using the `-n` option to `newfs`, the boot area will be present on your disk. If you create your file system using `mkfs` the boot area will not be present by default. In this case, if you will use the file system on this disk to boot your system, you must explicitly put the boot area on your disk using the following command (replace `0s0` for your disk's actual device file name):

```
dd if=/etc/boot of=/dev/dsk/0s0 count=1 bs=8k
```

Each boot disk must have a **volume header** in the boot area to identify the volume format (in this case, LIF). The volume header is checked by the boot ROM in its examination of bootable mass storage media when the computer is powered up.

The **volume directory information** contains SYSHPUX, SYSBCKUP, and SYSDEBUG. All three are object files.

SYSHPUX corresponds to the file `/hp-ux`, which is your kernel. SYSBCKUP corresponds to `/SYSBCKUP`, which is used as a backup kernel. SYSDEBUG corresponds to the file `/SYSDEBUG`. This file is used only when writing device drivers; its use is described in *Series 300 HP-UX Driver Development Guide*.

On Series 300/400, the last 7.25 KB on the boot area contain the **secondary loader**. The boot ROM loads and passes control to the secondary loader which in turn loads and passes control to the file `/hp-ux` (or the backup kernel if you are using SYSBCKUP). `/hp-ux` (or the backup kernel) then completes the task of bringing up HP-UX.

Series 700 Boot Area Implementation

Whereas the Series 300/400 has three locations hard-coded into the system, the Series 700 has an additional loader, which understands the layout of the file system. The Series 700 boot area has pointers to the actual bootstrap programs.

The `lifls` command on a Series 700 reports presence of `FS`, `SWAP`, `ISL`, `AUTO`, `HPUX`, `IOMAP`, `EST`, and `PAD` files. Its reportage of `FS` and `SWAP` indicates that Series 700 LIF has knowledge of the entire disk, including the file system and swap.

When the system is booted, the loader goes off and can find `/hpux` at a default or designated location, using the boot console user interface.

For more information, see the owner's guide for the Series 700 systems or *hpux_700*(1M) in the *HP-UX Reference*.

Series 800 Boot Area Implementation

On Series 800, the LIF header contains `ISL`, `HPUX`, `AUTO`, `RDB`, and `IOMAP` files. `ISL` uses the `AUTO` file to locate the HP-UX kernel.

Primary Swap Area

The primary swap area is the *first* swap device specified in the `S800` or `dfile`; it is also specified in `/etc/checklist`. The primary swap area is a contiguous area of the root disk used by the virtual memory system (see Chapter 7, "Memory Management") to temporarily store a process image. Until `/etc/rc` executes `swapon`, primary swap is your only swap device.

As device swap space (in comparison to file-system swap), primary swap is fast; the system accesses it directly, without going through a file system.

On Series 300/400/700 systems, the primary swap area occupies blocks after a file system area or an entire disk dedicated as a swap disk. If you have multiple disks, each one can contain its own swap area, but there is still only one primary swap area on the entire system.

On Series 800 systems using traditional disk sections, primary swap space occupies its own section, separate from the file system and boot area sections. Like Series 300/400, there can be a boot area section on each physical disk.

Although a single disk can have multiple swap sections, it is *not recommended* because performance will degrade as the system attempts to do interleaved writes to swap areas on separate areas of the disk. Instead, configure multiple swap areas on separate disks. (For discussion of interleaving, see Chapter 7, "Memory Management".)

On Series 800 systems using LVM, the primary swap area resides in the root volume group in a designated logical volume. As with a disk-section configuration, you can set up multiple swap areas in logical volumes that are on separate disks (physical volumes).

You can list all swap areas on your system using the *swapinfo*(1M) command; see the *HP-UX Reference Manual* for complete specifications.

Procedures for managing primary swap space (as well as secondary swap) are found in two chapters of the *System Administration Tasks* manual: Chapter 6, "Managing Swap Space" and "Managing Logical Volumes."

File System Layout

With the exception of disk drives used for raw data, every disk drive contains some file systems. All HFS file systems are laid out in a common format, with the following structures:

- Primary superblock
- Multiple cylinder groups

The many data structures governing the superblock and cylinder group are defined in several header files, particularly `/usr/include/sys/fs.h`. Superblock headers, defined in `fs.h`, also include absolute disk addresses for the first boot block and definitions of numerous file system attributes, including cylinder-group characteristics (such as rotational positions, number of inodes per group, number fragments per block), file length, and mirror states of root and primary swap.

A description of the file-system format is found on *fs*(4) of the *HP-UX Reference Manual*.

The Superblock

The **superblock** is a contiguous, 8-KB block of disk space near the beginning of the file system's disk section. The superblock contains a record of the *static* information about the state of the file system at the time of its creation (or extension, if using LVM):

- file system size
- number of inodes it can store
- locations of free space on the file system
- number of cylinder groups
- location of superblocks, cylinder groups, inode blocks, and data blocks
- size and number of blocks and fragments.

The primary superblock also keeps track of file system update information in its summary information area.

HP-UX uses information in the superblock for various file system maintenance procedures—for example, when you mount a file system or perform a file system check by executing `fsck`.

Because the superblock is so important, HP-UX always keeps redundant copies on disk in each cylinder group. One copy is brought into main memory when you boot up. A primary superblock is at the beginning of the file system, and each cylinder group has a copy of the superblock. This redundancy further ensures the integrity of file system data.

The non-redundant superblocks on the disk are updated whenever the `sync` command is executed and when a file system is unmounted (see *sync*(1M) in the *HP-UX Reference*).

Record of all superblock locations can be found in two files—`/etc/sbtab` for Series 300/400/700 and `/etc/super_blocks` for the Series 800.

The Cylinder Group

The cylinder group is the term used to describe a further internal organization of disk sections.

Here is the principle, illustrated in Figure 8-6—A cylinder is a collection of tracks located the same distance from the edge of a disk platter. Since all the tracks in a cylinder are accessed by the read/write heads of the disk drive simultaneously, the blocks of space on each track can be accessible with minimum rotational latency; that is, requiring no seek time.

For performance reasons, small groups of adjacent cylinders (sixteen by default, see *newfs*(1M)) are grouped together as **cylinder groups**. Each cylinder group has its own set of inodes and local mappings of free space in the group. This internal organization results in both bringing to closer proximity file-system inodes and their associated data without long seeks and dispersing data and inodes across cylinders. Minimum time is lost seeking file data within a cylinder group.

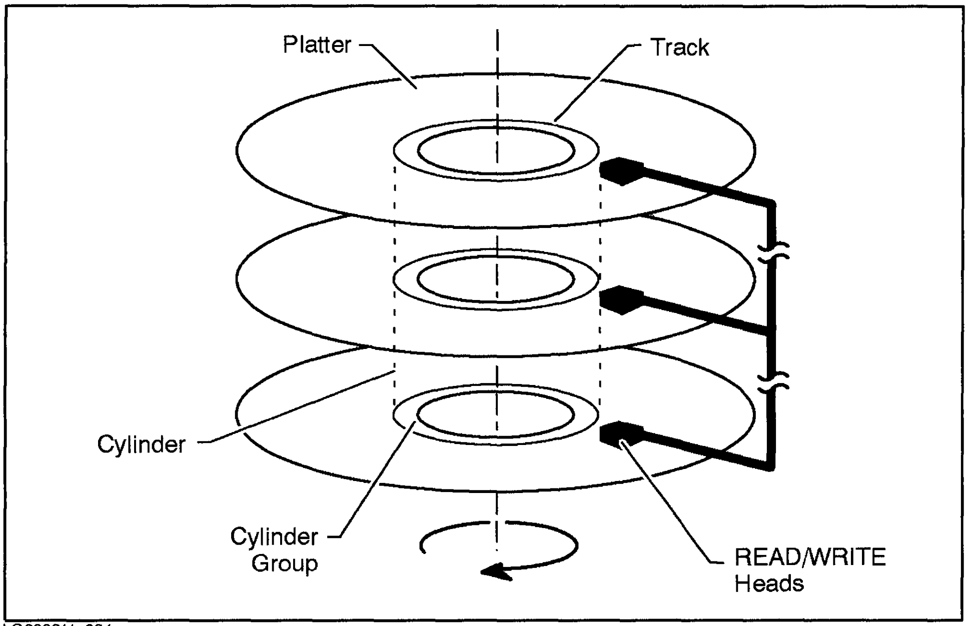


Figure 8-6. Track, Cylinder, and Cylinder Group on a Disk

The cylinder group controls all access to a file and its associated data. Each cylinder group contains a copy of the superblock, a cylinder group information structure, an inode table, and data blocks (see Table 8-4).

8

Table 8-4. Cylinder Group Layout

Data Structure	Size
Boot block[1]	8 KB
Primary superblock[1]	8 KB
Redundant superblock	8 KB
Cylinder group information	1 block (4 or 8 KB)
Inode table	varies[2]
Data blocks	0 or more blocks[3]

1 First cylinder group only. The beginning of all subsequent cylinder groups might be filled by data blocks, depending on offset.

2 See "Inodes" section.

3 Due to offset; see "Data Blocks" section.

A redundant copy of the superblock is located in each cylinder group. This ensures that if any single track, cylinder, or platter is damaged, the file system itself can be repaired by executing `fsck` and specifying an alternate superblock. Further, each successive cylinder group is laid out offset by one track in relation to the previous cylinder group, so that the redundant copies of the superblock spiral down the platters.

The **cylinder group information** contains the *dynamic* parameters of the cylinder group:

- Number of inodes and data blocks in the cylinder group
- Pointers to the last used block, fragment, and inode
- Number of available fragments
- Used inode map
- Free block map.

The cylinder group information data structure's size is one block (a block can be defined when running `newfs` as either 4 KB or 8 KB). The layout of the cylinder group information is defined in `/usr/include/sys/fs.h`.

Inodes

Besides maintaining information about the state of the file system, the cylinder group also maintains key information about the inodes of the file system—the system's index to the actual files of data. Inodes contain the locations of the actual file data.

The cylinder group maintains an **inode table**, which provides summary information about each file in the cylinder group (see Figure 8-7). In addition, the "disk inodes" appear in an expanded version ("in-core inodes") in memory for inodes currently (or recently) used.

A disk inode includes the following information:

- mode and file type
- number of links to the file
- owner and group information
- file size in bytes
- time stamps
- pointers to the file's actual blocks of data on disk

When a file is read into memory, its "in-core inode" contains the following additional fields:

- status of the in-core inode, including if the inode is locked, if a process is waiting for the inode, if the disk inode now differs from the in-core copy due to file modification, if the file is a mount point
- numeric address of the file system containing the file
- inode array number by which the kernel identifies the disk inode
- pointers to other in-core inodes linked on buffer hash list and free list

The header file, `/usr/include/sys/inode.h`, defines the in-core inode; the header file, `/usr/include/sys/ino.h`, defines the disk inode.

When the operating system accesses a file, it finds the file by means of the inode's pointers to the file's blocks of data. The scheme for finding the file data is discussed later in this chapter. 8

A static number of inodes is allocated for each cylinder group when the file system is created. HFS uses a default that provides sufficient inodes per cylinder group for average usage.

If the file described by the inode is not a regular file, some of the inode fields differ as follows:

- FIFO and pipes
  The space reserved for indirect block pointers contains information about the current state of a FIFO or pipe.
- Character or block device files
  The first direct block address is actually the major and minor number of the device. The rest of the direct block addresses are 0.
- Directories
  The pointers point to regular file system data blocks, but the blocks contain specifically formatted data, described in the header file `/usr/include/sys/dir.h`.

When you create a file system (using `newfs` or `mkfs`), the system creates inodes. The number of created inodes limits the number of files that you can have in a file system. Each time you create a file, an inode is allocated for that file. Both commands default to 2048 bytes per inode, meaning the system assumes that the average size of your files will be 2048 bytes.

Although uncommon, an inode error message, `inode: table is full`, might require changing the size of the inode table. This message refers to the kernel's in-core inode parameter. A configurable parameter, `ninode`, defines the maximum number of open, in-core inodes.

You can use SAM to change these configurable parameters, which are documented in *System Administration Tasks* manual for your system.

Data Blocks

Disk space before or after the superblock, cylinder group information, and inode table is filled with data blocks. (The specific locations of data on each platter is different, due to the cylinder-block offset.) The blocks are used to store data for regular files, directories, and symbolic links.

HP-UX provides support for file systems in several block sizes:

- On the Series 300/400/800, the file system can use blocks of either 4 KB or 8 KB.
- On the Series 700, the file system can use block of 4 KB, 8 KB, 16 KB, 32 KB, or 64 KB.

Block size is set using the `mkfs` or `newfs` command, when you construct a file system. See *mkfs*(1M) and *newfs*(1M) in the *HP-UX Reference* for details.

Larger block sizes are faster for sequential access to the file system, while smaller block sizes use space more efficiently and are better for random I/O. Having a large block size has both benefits and costs. For big files, a large block size significantly reduces the number of disk accesses, thereby increasing file system throughput. The problem is that most HP-UX files are small; thus, using a large block size for small files might waste space.

In the `fs.h` header file, the size of blocks is referred to as `fs_bsize`, depending upon what block size your file system uses.

To minimize wasted space, a block can be divided into either 1 KB, 2 KB, or 4 KB fragments.

Fragment size is bounded on the lower end by 1024 and on the upper end by the block size. Fragment size is specified at file system creation, can never be less than one-eighth of a block, and must be one, two, four, or eight times 1024.

How a File is Accessed from Inode to Data Blocks

As described earlier, the inode in the cylinder group contains pointers to the locations of a file's actual data. Depending on the size of a file, its data might be reached through pointers to direct blocks or indirect blocks, which are pointers to a block containing more pointers to the data. HP-UX allows for up to triple indirect pointing for enormous files.

Figure 8-7 shows the mapping from an inode to a file's data blocks.

The first 12 pointers in an inode point directly to the first 12 blocks or fragments containing the file's data. If the file is larger than 12 blocks (greater than 12 $\times$ *fs_bsize*), indirect reference is made to the file's data. A group of indirect pointers is contained in one data block. Each pointer is 4 bytes long, so there can be either 1024 pointers (4096/4) or 2048 pointers (8192/4) in each block of indirect pointers.

The thirteenth block address in the inode points to a block containing 1024 or 2048 additional pointers to data blocks (from now on, the number of indirect pointers in a block will be called *num_ip*). Thus, the thirteenth (single indirect) block address handles files up to 4,243,456 bytes in a 4-KB block file system or 16,875,520 bytes in an 8-KB block file system ($fs\_bsize \times (12 + num\_ip)$) . If the file is larger than this, the fourteenth inode block address points to *num_ip* indirect blocks, each of which contains pointers to an additional *num_ip* actual data blocks.

If the file cannot be contained in this space, the fifteenth inode block address points to *num_ip* double-indirect blocks. With the fifteenth (triple-indirect) block address, the size of a file is limited to $fs\_bsize \times (12 + num\_ip + num\_ip^2 + num\_ip^3)$.

8

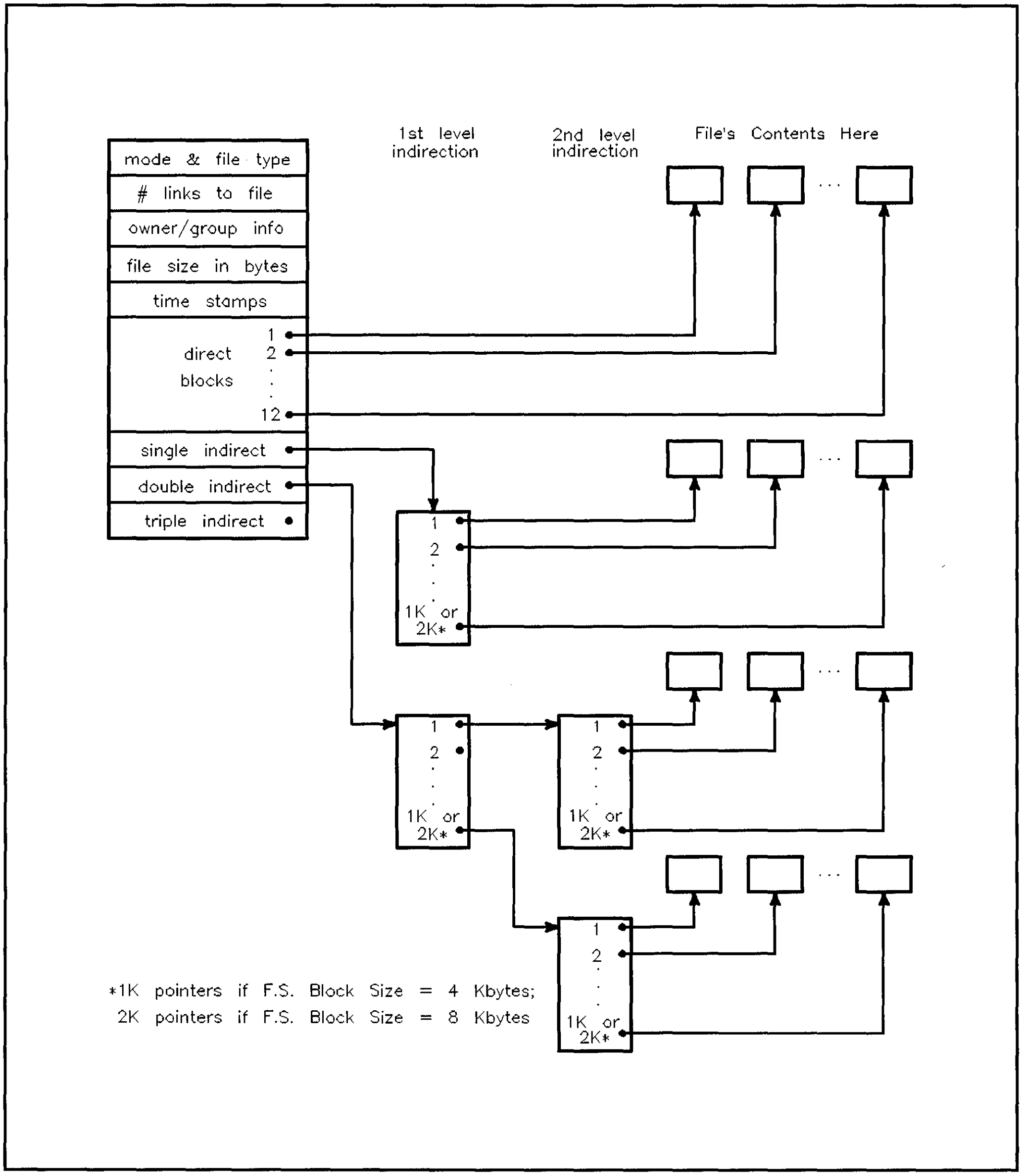


Figure 8-7. Mapping from Inode to File Data Blocks

Inode pointers hold the address of a fragment. The address can be interpreted as referencing an entire block or one or more fragments, depending on the number of bytes stored at the address.

All blocks but the last have a full block of data allocated to them. If the amount of data in the last block is less than the file system block size, only the number of consecutive fragments needed to actually store the actual data are allocated. For example, in an 8-KB/1-KB file system, a 15-KB file is stored as 2 8-KB blocks and 3 consecutive 1-KB fragments. (The latter might also be referred to as a 3-KB fragment.) This allocation scheme provides the performance advantage of large blocks with the space savings of small fragments.

Figure 8-8 shows an example of a 20-KB file stored in 8-KB blocks with 1-KB fragments. The number of blocks needed is 20 ÷ 8 (file size ÷ block size): 2 full blocks with a remainder of 4 fragments. Therefore, the first and second pointers point to full blocks, but the third pointer points to the remaining 4 fragments.

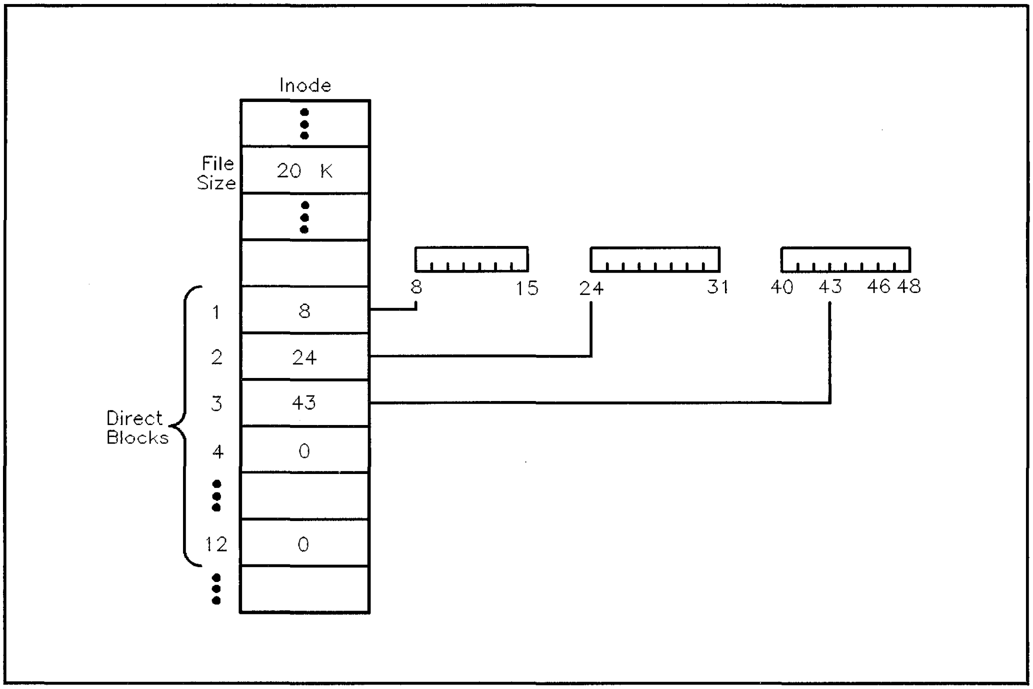


Figure 8-8. Inode Addressing Example

All indirect blocks are referenced only as full blocks; no pieces of the file are addressed at the fragment level beyond the 12 direct pointers.

When a block or fragment is needed, the disk is searched for free blocks. Ideally, free blocks should be found throughout the disk, for searches to locate a free block close to related blocks. When the file system is full, there are long linear searches to find the block, and when a block is allocated, it is likely to be placed far from the previous block of the file, resulting in long seeks and slow performance.

8

Minimum Free Space

To ensure the availability of free blocks near one another, a certain percentage of free space must always be available in the file system. This minimum free space percentage is specified at file system creation using the `-m` option of the `newfs` command or the `minfree` argument of the `mkfs` command. The default is 10 percent. Values lower than 10 percent may severely degrade system performance, by causing the file system to search harder for free space.

The percent of free space can be changed at any time using the `-m` option of the `tunefs` command. The reserved free space is inaccessible to the normal user; once this threshold is met, only the superuser can continue to allocate blocks. When the percentage of free space drops below the threshold, system throughput (to and from newly created files) drops because the file system can no longer localize the blocks for a file. Accessing a file is quicker if the whole file is grouped together (localized).

How Disk Space is Allocated

Free space availability is determined from a bit map associated with each cylinder group. The bit map contains one bit for each fragment. To determine if a block is available, the system examines consecutive fragments. A piece of the bit map from a file system using 1024-byte fragments and 8192-byte blocks is shown in Table 8-5.

Table 8-5. Sample Free Block Bitmap in an 8KB/1KB File System

bit map	00000000	00000011	11111100	11111111
Fragment numbers	0-7	8-15	16-23	24-31
Block numbers	0	1	2	3

Fragment numbers 14-21 and 24-31 in this example are free, indicated by *1*s in the bit map. Fragment numbers 0-13 and 22-23 are allocated, as indicated by *0*s in the bit map. Fragments in adjacent blocks cannot be used to create a full block; only eight contiguous fragments starting on a block boundary can be used to allocate a full block. Fragments 24-31 can be coalesced to form a full block, but not fragments 14-21. Also, if a partial block is allocated, the fragments must be consecutive and not cross a block boundary. For example, if three fragments are needed, fragments 16-18 can be allocated, but not fragments 14-16.

Every time data is written to an existing file, the system checks to see if file size must increase. If so, one of three conditions exists:

- Sufficient space exists in the existing block or fragment; the new data is written into the already allocated space.
- The file contains only whole blocks; the last block contains insufficient space to hold additional data. If more than a full block of data must be written, a new block is allocated and written. This process is repeated until less than a full block of new data is needed. At that point, a block containing enough contiguous fragments is located and the new data is written there.
- The file contains fragments, but not enough to hold the new data. If the size of the existing data in fragments plus the new data exceeds the size of a full block, a new block is allocated. Both the old and new data are written to

the new block. If the size of the old and new data is less than a full block, a block with enough contiguous fragments (or a full block) is located and allocated.

When a block or fragment has been located, the address is recorded in the inode table and the free block bit map is updated.

Allocation Policies

Allocation is performed globally to place new directories and files and locally to place data in blocks.

A global decision determines which cylinder group will contain a given file or directory. HP-UX attempts to put all files from a single directory in the same cylinder group. Newly created directories are put in the cylinder group with the greatest number of free inodes and the smallest number of directories.

Global policy specifies that once the file size reaches `maxbpg` (`maxbpg` is defined via the `tunefs` command), HP-UX allocates blocks from another cylinder group. This helps to enforce grouping of all files within one directory into a single cylinder group by spreading the less common larger files over several cylinder groups.

Global allocation routines call local allocation routines with requests for specific data blocks. Blocks are allocated by the following priorities:

- Allocate block requested.
- Allocate a block on the same cylinder that is rotationally closest to the requested block.
- Allocate any block within the same cylinder group.
- Use a quadratic hash to find a new cylinder group; allocate a block somewhere in the new cylinder group.
- Use sequential search to find an available block.

Speed in allocating blocks is the most important characteristic of this strategy. For this reason, the percentage of free space must be maintained.

The File-System Buffer Cache

The file-system buffer cache manages data flow between main memory and secondary memory (principally disks), by temporarily holding (buffering) information about data being transferred to and from disk. The buffer cache speeds data transfer from the file system to main memory; once buffered, data is accessed by a process's executing space in main memory much faster than from the file system on disk. The buffer cache is used for all file system I/O operations, plus all other block I/O operations in the system (for example, `mount`, inode reading, LVM management, and some device drivers).

The role of the buffer cache is illustrated in Figure 8-9. When you execute a program, the shell passes the file path name to `exec`, finds the file on disk, and reads the `a.out` header into the buffer cache. The `a.out` header contains preliminary information about the executable, including the size of the text and the uninitialized data (bss areas).

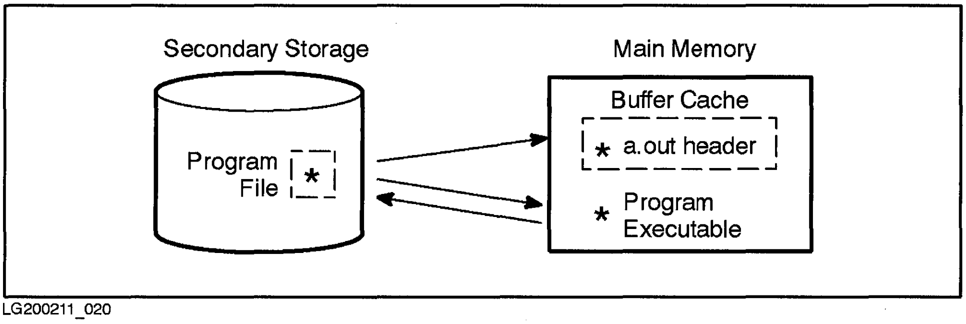


Figure 8-9. Buffer Cache Holds the a.out Header of Executing Programs

As the code executes, the virtual-memory system reads the pages of data directly from the disk into memory. (Some additional pages might also be read in, based on the probability that they will be needed). The file's `a.out`, which is only needed to begin the "demand-paging", might (or might not) remain in the buffer cache throughout the process execution, depending on whether its buffer is needed.

If you have just created and compiled a program, all transactions occur from the buffer cache. For an existing program, however, data might exist on both disk and buffer cache. When a page is faulted in from disk to memory, HP-UX also ensures that the process executes using the most current copy of the data. During a file-system write, HP-UX ensures that only the most current copy of the data, whether in the virtual-memory system's page cache or in the buffer cache, is written to disk.

The HP-UX file-system buffer cache is currently implemented in two ways:

- On Series 300/400/700 systems, the buffer cache is dynamic; that is, it can change in size depending on system demand for virtual memory vs. buffer cache. (This dynamic capability is described in the next section.)

  On HP-UX clusters, the `dskless_fsbufs` parameter governs the maximum number of buffers that you can create for incoming cluster traffic. `dskless_fsbufs` is a separate pool of buffers, which are allocated as needed. (Clusters are not supported for Series 800 systems in the HP-UX 9.0 release.)
- On the Series 800, the number of buffers in the cache is set by two operating-system parameters in the `S800` file—`nbuf` and `bufpages`. When you power up your system, these parameters reserve memory for buffer headers (`nbuf`) and for pages of memory for buffer-cache use (`bufpages`) based upon the amount of available RAM. Of the two parameters, `bufpages` is more critical, defining the amount of memory in buffer cache, which can vary depending on block size.

  You can use SAM to change the default buffer-cache values and then reboot to implement the changes. For information on `nbuf` and `bufpages`, refer to the *System Administration Tasks* manual.

The buffer cache consists of two parts:

- buffer headers, which have pointers to the buffer and describe its contents.
- buffer data area, which reside in data blocks ranging in size from `DEV_BSIZE` to `MAXBSIZE`.

Just as a file system must be created in a multiple of `DEV_BSIZE`, so must a buffer always be some multiple of `DEV_BSIZE`. `MAX_BSIZE` is the largest buffer size, in bytes. The smallest unit of memory assigned to a buffer is one page.

The data structures used for allocating and managing buffers are defined in the `/usr/include/sys/buf.h` header file.

Requests for buffers come from many sources, including file-system reads and writes, and device driver allocations. If a buffer is requested and not already in the cache, the operating system obtains the buffer header, allocates memory for the pages of the buffer, and then gives it to the part of the operating system making the request.

In the standard (non-dynamic) implementation, the buffer headers are allocated in a single contiguous block and treated as an array. Inactive buffer headers are placed on one of three doubly-linked lists—LRU (least recently used), AGE, and EMPTY:

- Although its name suggests otherwise, the LRU list actually points to blocks of most frequently accessed data, representing no more than 40% of total buffers. If data in a buffer is dirty (that is, its contents changed since accessed from the file system), its pages must be written to disk before the pages can be reallocated.
- The AGE list contains buffers accessed less frequently and the overflow of the LRU list.
- The EMPTY list contains unallocated buffer headers.

If a buffer requires more than one page, HP-UX ensures that the pages are assigned in consecutive addresses.

As code and data move into the buffer cache, the system copies the information from the buffer cache into user's main memory. If a user requests information already in the buffer cache, the information is copied from the cache to user's main memory, eliminating the I/O operation to bring it in from the file system disk.

When data is written through the buffer cache, any data in the virtual-memory system's page cache (in main memory) with the same `vnode` and block address is purged. Virtual addresses used by the buffer cache are in kernel space. When a pagein occurs, both the buffer header (on one of the buffer lists) and associated data in the buffer cache are flushed.

Dynamic Buffer Cache (Series 300, 400, and 700 only)

The buffer cache is implemented on Series 300/400/700 systems to grow and shrink in size, depending on operating-system and virtual-memory need. Note that demand for memory is generated not only by the file system, but also by other objects, including processes, data regions, memory-mapped files.

From a system-administration perspective, using the dynamic buffer cache is simple: the operating system is shipped with it set up. The `nbuf` and `bufpages` operating-system parameters are not specified in the `dfile`; in their absence, operating system determines how many buffer-cache pages are needed for optimal system performance. You can choose to configure these parameters, but if you do, the buffer cache will no longer function dynamically.

Both buffer cache and virtual-memory subsystem access the same body of RAM in main memory. The dynamic buffer cache is allowed to grow considerably larger than a statically configured buffer cache, permitting more data to be held in memory. When the virtual-memory system requires more memory, the dynamic buffer cache is reduced to yield memory for processes.

The dynamic buffer cache functions like a large memory-mapped file that is shared among all the processes running on the system. (Note that there are a number of subtle interactions between the buffer cache and memory-mapped files, which can streamline bringing data into the virtual-memory subsystem. For information on memory-mapped files, see Chapter 7, "Memory Management".)

The dynamic buffer cache uses a different algorithm for reusing existing buffer pages and allocating more pages from memory. It still reuses pages according to "least-recently used" (LRU) order. However, instead of maintaining three free lists (LRU, AGE, and EMPTY) as in the standard buffer-cache implementation, the dynamic buffer cache uses two free lists (LRU and EMPTY).

The LRU lists buffers in most-recently to least-recently used order. This list may grow as long as the buffer cache is growing. When a buffer is read for the first time, its buffer is inserted mid-list, in fairly high priority. If accessed again, its priority is increased. Other buffers might decrease in priority (such as file-system writes to an entire block, which typically do not get referenced again).

The dynamic buffer cache shrinks by use of `vhand`, the virtual-memory subsystem's pageout daemon. (See "Maintaining Page Availability—vhand and swapper Daemons" in Chapter 7, "Memory Management.")

`vhand` reclaims pages of memory from the buffer cache as well as virtual memory, by using reference bits, much as it does through the virtual memory subsystem's regions. Its first (age) hand clears the status bits of any buffer pages not recently accessed. If the status bit remains clear by the time the second (steal) hand traverses it, `vhand` reclaims (pages out) the associated page.

The dynamic buffer cache provides the operating system flexibility to accommodate both small application programs that do a lot of I/O and large programs that do little I/O but require many pages of memory for data.

How the HFS File System Modifies Files

Every time a file is modified, the HP-UX operating system updates the file system to ensure its consistency.

When a process updates (writes to) the file system, the data being written is copied into an in-memory buffer cache. The physical disk is updated asynchronously from the buffer write. The data, along with the inode information reflecting the change, is written to the disk sometime later, unless the file was opened in the synchronous mode (see the description of `O_SYNC` and `O_SYNCIO` in *open*(2) and *fcntl*(2) in the *HP-UX Reference*). The process continues, even though the data has not yet been written to the disk. If the system is halted without writing the buffer to disk, the file system on the disk is left in an **inconsistent state**. Such inconsistencies are flagged and corrected, if possible, by the `fsck` command at system startup. (Discussions of `fsck`, the file-system check command, appear later in this chapter, in *Solving HP-UX Problems*, and in *fsck*(1M) of the *HP-UX Reference Manual*.)

The `sync` command can be used to force synchronization. However, the `syncer` command routinely updates the file system's superblock, inodes, data blocks, and cylinder group information, as described below. (For further information, see *sync*(1M) and *syncer*(1M) in *HP-UX Reference Manual*.)

Primary Superblock	The superblock of a mounted file system is written to the disk whenever a `umount` command is issued, or when a `sync` command is issued and the file system has been modified.
Inodes	An inode contains information describing the file. The inode is written to disk after every modification, unless the `fs_async` parameter is set in the configuration (`S800` or `dfile`) file. (See "Synchronous vs. Asynchronous Disk Writes", later in this chapter.)
Data blocks	In-core blocks (including directories, indirect blocks, files, pipes, symbolic links, and FIFOs) are written to the file system after being modified and released by the operating system. Upon release, data blocks are buffered or queued for eventual writing. Physical I/O takes place when the buffer is needed by HP-UX, when a `sync` or `fsync` command is issued, or when `O_SYNC` is set for the file. If a file is opened with the `O_SYNC` or `O_SYNCIO` flag set, the `write` system call does not return until completed.
Cylinder group information	The cylinder group information is updated whenever a `sync` is executed, or when the system needs a buffer and the cylinder group is written.

Caution

- Always unmount a file system *before* executing `fsck`.
- Always reboot the system *without* `sync`ing (that is, use `reboot -n`) after altering the root device with `fsck`.

A file system can become inconsistent if you execute `fsck` on a mounted file system other than the root file system; you risk missing buffered information not yet written to the file system. If this information is then flushed from the buffer cache, it might overwrite corrections that `fsck` had made.

8

Immediate Reporting

Numerous SCSI disk devices are shipped with a feature called immediate reporting (disabled by default). Immediate reporting speeds status notification; its implementation is handled by the disk controller and disk device. However, immediate reporting also has some associated risks.

With immediate reporting, when a device driver sends a write request to a device, the device accepts the data, places it in its buffer or its cache, and reports to the SPU that the write completed successfully. Without immediate reporting, status is not returned until the data goes to the media itself.

In a power (or other) failure, data might not have been written successfully to disk, but in fact, still reside in a buffer. An application, writing to the raw device or to the files system using `O_SYNC`, continues processing as though the data has been written. If data remains in the buffer at the time of a system failure, the database is left in an inconsistent state.

Under rare circumstances, immediate reporting might also cause delayed errors or system panics. This can occur in the following scenario: A user has a write request and the system returns good status immediately. If the next request is a kernel request and an error occurs (such as a write failure) caused by the user's write request, the error might get associated with the kernel request. If the kernel request cannot tolerate the error, the kernel might panic.

Immediate reporting can be set or disabled using the *scsictl*(1M) command. If it is critical that your system never go down, you might want to disable immediate reporting. Although SCSI disks available for Series 800 systems can be set for immediate reporting, the feature poses greater risk of inconsistent data; the disks are shipped with the feature disabled.

Synchronous vs. Asynchronous Disk Writes

When HP-UX writes a file-system data structure to disk synchronously, any file-system activity must complete to the disk before the program is allowed to continue; the process does not regain control until completion of the physical I/O (regardless of whether the I/O is user data or operating-system data). Synchronous writes include some file-system structures and whatever an application writes with `O_SYNC` set.

When HP-UX writes to disk asychronously, I/O is scheduled at some later time and the process regains control immediately, without waiting.

By default, some critical changes to the structure of the file system are posted to disk synchronously. Synchronous writes ensure file system integrity in case of system crash, but this kind of disk writing also impedes system performance. Run-time performance increases significantly (up to roughly ten percent) on I/O-intensive applications when all disk writes occur asynchronously; little effect is seen for compute-bound processes. However, if a system using asynchronous disk writes crashes, recovery might require system-administrator intervention using `fsck` and might also cause user data or directories to disappear.

As a system administrator, you can specify whether some disk writes are performed synchronously or asynchronously. The `fs_async` parameter in the `S800` (Series 800) or `dfile` (Series 300/400/700) enables and disables the feature regarding inodes. (You cannot modify whether or not other types of disk writes occur synchronously. They are asynchronous by default and synchronous if `O_SYNC` flag is set by the application.)

- On the Series 300/400/800, the `fs_async` default value of `0` specifies that the writes should be performed synchronously. Setting `fs_async` to `1` causes fewer writes to be performed synchronously. Typically, this causes file-system performance to improve.
- On Series 700 systems only, the default value of `0` specifies that writes be performed asynchronously.

Note too, `fs_async`, deals with inodes and directories, while `O_SYNC` deals with files and data. If a file is opened via `O_SYNC`, the file continues to be written synchronously, regardless of what method is specified. `O_SYNC` also causes inodes to be updated synchronously.

Although asynchronous disk writes increases system performance for most applications, if a system crashes, file-system data structures are likely to be left in an inconsistent state. For this reason, we do *not* recommend that you turn on `fs_async` on a production system.

Normally, file-system recovery is performed automatically by `fsck` in the reboot process and does not require any intervention by the system administrator. However, using asynchronous disk writes might require system administrator intervention in the event of a crash. For further information, refer to *fsck*(1M) in the *HP-UX Reference*.

Minimizing File-System Corruption

Although the HFS file system is very reliable, hardware failures, accidental power loss, or improper shutdown procedures can cause its corruption.

Problems, such as a bad block on a disk, power loss, or a non-functional disk controller, can occur and cause the hardware to fail. By following recommended hardware preventive maintenance procedures and by keeping regular backups (as defined in the *System Administration Tasks* manual), you can avoid most serious problems and be prepared for any that might occur.

As a system administrator, you are responsible for preserving users' data. Since the file system is the HP-UX data structure that stores the data, it is essential that you safeguard the file system by performing maintenance tasks (such as regular backups), following proper startup and shutdown procedures, and by checking the file system when necessary using the `fsck` command. (*fsck*(1M) is discussed shortly, in this chapter, in *Solving HP-UX Problems*, and in *HP-UX Reference Manual*.)

System Shutdown and Startup Guidelines

To ensure file system integrity, always follow proper shutdown and startup procedures (described in the *System Administration Tasks* manual):

- Always shut down the system using `reboot` or `shutdown`.
- Never physically write-protect a mounted file system, unless it is mounted read-only.

- Never take a mounted file system off-line (for example, by shutting its power off or by disconnecting it) while it is in use.

Follow proper startup procedures (described in the *System Administration Tasks* manual):

- Always check the file system for inconsistencies. (The `fsck` command runs automatically when the system reboots.)
- Always repair inconsistencies, using `fsck`. Allowing a corrupted file system to be further modified in such circumstances can be disastrous.

The /lost+found Directory

Every file system should have a `lost+found` directory at its root. `fsck`, the file system check command (discussed in the next section), places any problem files or directories in `lost+found`. After `fsck` completes, you should examine each file in `lost+found`, to determine its name and location, and attempt to return it to its rightful place.

`lost+found` is created by both `mkfs` and `newfs` when they create file systems. However, if your system lacks `lost+found`, you can rebuild it using *mklost+found*(1M). `mklost+found` creates several empty file slots for `fsck`.

Understanding Use of fsck to Detect and Correct File-System Corruption

The `fsck` command is the principal file-system maintenance tool for checking system consistency and making repairs.

8

Never run fsck on a mounted file system. However, `fsck` should be run regularly to ensure the file system's structural integrity:

- `fsck` is invoked during system boot-up by the `/etc/bcheckrc` script run by `init`.
- For preventative maintenance, `fsck` should be run weekly (before each full backup) on all file systems, but particularly on file systems that have been unmounted.
- You should run `fsck` *any* time you suspect problems with the HP-UX file system. Be sure to unmount the file system first!

In performing its checks, `fsck` examines the file system several times, each time examining different characteristics, including:

- Block and file size
- Path names
- Connectivity (parent-child relationships)
- Reference count links
- Cylinder groups

`fsck` checks intrinsically redundant file-system data. The redundant data is either read from the file system or computed from known values.

The file system should be in a unmounted state when you check it. The root file system should *only* be run from `init` run-level `s`, the system administrator run-level. (Thus, you can check the root file system after performing a system shutdown.) Do not run `fsck` for the root file system when the system is busy. You can check non-root file systems any time, but be sure they are unmounted.

You can run `fsck` interactively or non-interactively. When invoked without options, `fsck` runs interactively on file systems marked "hfs" in `/etc/checklist` and queries you for a response when it finds an inconsistency. In non-interactive mode (typically, in the `-p` or preen mode), `fsck` reports inconsistencies, corrects many problems, but does not remove data. If it cannot solve a problem, `fsck` terminates. If this happens, you should run `fsck` interactively to fix the problems.

Note When running `fsck -p` before a backup, if the command completes successfully, perform your backup. If it aborts with errors, back up the bad file system, repair it, then back up the file system again.

Do not issue the `reboot` command in its default form after `fsck` has repaired a mounted file system. By default, `reboot` executes `sync` on the disks, thus writing out inconsistent data. If you must reboot, use `reboot -n`, which does not issue a `sync`.

Refer to *Solving HP-UX Problems* (Chapters 6 and Appendix A) for full discussion of `fsck` options and actions. The following subsections describe the interaction of `fsck` on various elements of the file-system.

Superblock Consistency

The superblock's summary information can become inconsistent because every change to the file system's blocks or inodes modifies it. Most often, the superblock and its associated parts become corrupted when the computer is halted and the last command involving output to the file system is not a `reboot`, `shutdown`, `sync`, or `umount` command. `fsck` checks the superblock for inconsistencies involving:

- Free block count—this is fairly common
- Free inode count—this is fairly common
- File system size—this rarely happens.

If `fsck` detects corruption in the static parameters of the primary (default) superblock, `fsck` requests the system administrator to specify the location of an alternate superblock. The alternate superblock addresses are listed during file-system creation. An alternate superblock is always found at block number 16. If this superblock is also corrupted, you must supply the address of another superblock. If the last time you created a file system was during the installation, a list of superblock addresses can be found in the `/etc/sbtab` file.

File System Size

`fsck` examines the superblock for inconsistencies involving file system size, number of inodes, free block count, and the free inode count. The file system size must be larger than the number of blocks used by the superblock and the number of blocks used by the list of inodes. The file system size and layout information are critical pieces of information to the `fsck` program. While there is no way to actually check these sizes, `fsck` can verify that they are within reasonable bounds. All other checks of the file system depend on the correctness of these sizes.

8

Free-Block Checking

`fsck` checks that all data associated with files and directories can be found.

The superblock summary information contains a count of the total number of free blocks within the file system. `fsck` checks that all the blocks marked as free are not claimed by any files. When all the blocks have been accounted for, `fsck` compares this count to the number of free blocks it finds within the file system. If the figures do not agree, `fsck` replaces the count in the summary information by the actual free-block count.

If anything is wrong with the free-block maps, `fsck` rebuilds them, excluding all blocks in the list of allocated blocks.

Inode Checking

The superblock summary information contains a count of the total number of free inodes within the file system. `fsck` compares this count to the number of free inodes it finds within the file system. If the figures do not agree, `fsck` replaces the count in the summary information by the actual free inode count.

Inode Consistency

Individual inodes are less likely than superblock summary information to be corrupted. However, because of the great number of active inodes, it is possible that a few inodes might become corrupted.

The inodes list is checked sequentially, from inode 2 (inode 0 marks unused directory slots and inode 1 is reserved for future use) to the last inode in the file system.

The inode structure is defined in `/usr/include/sys/inode.h`. There are two major types of inodes: primary and continuation. Continuation inodes contain only a mode (which is of type continuation), a link count, and access control list (ACL) entries. Continuation inodes exist only if a file has optional ACL entries associated with it. `fsck` checks the continuation inode's mode, link count, and the reference from the primary inode. It does not examine the ACL information itself. `fsck` checks each primary inode for inconsistencies in the following areas:

- Format and type
- Link count
- Duplicate blocks
- Bad blocks
- Inode size
- Block count

Format and Type

`fsck` verifies inodes classifications—regular file, directory, block special file, character special file, network device, FIFO, symbolic link, or continuation inode. It also examines the inode state, as:

- Unallocated
- Allocated
- Neither unallocated nor allocated

This last state indicates an incorrectly formatted inode. An inode can get into this state, for example, if bad data is written into the inode list through a hardware failure. To correct such an ambiguous state, `fsck` clears the defective inode.

Link Count

Contained in each inode is a count of the total number of directory entries linked to the inode. `fsck` verifies the link count stored in each inode by traversing the total directory structure (starting from the root directory) and calculating an actual link count for each inode.

If the stored link count is non-zero and the actual link count is zero, no directory entry appears for the inode; `fsck` links the disconnected file to the `/lost+found` directory. If the stored and actual link counts are non-zero and unequal, a directory entry may have been added or removed without the inode being updated. `fsck` replaces the stored link count in the inode by the actual link count.

Duplicate Blocks

Duplicate blocks can occur from using a file system with blocks claimed by both the free-block list and other parts of the file system.

8

Each inode contains a list (or for large files, pointers to lists in indirect blocks) of all blocks containing its file's data. `fsck` compares each block number claimed by an inode to a list of allocated blocks. `fsck` updates the list of allocated blocks to include the block number. If a block number is already claimed by another inode, `fsck` adds the block number to a list of duplicate blocks.

To resolve duplicate blocks, `fsck` makes a partial second pass of the inode list to find the duplicated blocks' inodes. `fsck` prompts the operator to clear both inodes. Often clearing only one inode solves the problem, but the data in the other inode is suspect.

Bad Blocks

Contained in each inode is a list or pointer to lists of all the blocks claimed by the inode. `fsck` checks each block number claimed by an inode for a value outside the range of the file system (lower than that of the first data block or greater than the last block in the file system). If the block number is outside this range, the block number is a bad block number.

`fsck` prompts the operator to clear the inode.

Series 800 systems using LVM have another mechanism for relocating bad blocks. LVM bad block relocation is discussed in Chapter 9, "Logical Volume Manager".

Inode Size

Each inode contains a 64-bit (eight-byte) size field indicating the number of characters in the file associated with the inode. `fsck` uses the inode size field to check for size inconsistencies.

`fsck` calculates the number of blocks that should be claimed by an inode by dividing the number of characters in the file by the number of characters per block and rounding up to get the number of direct blocks. `fsck` then counts actual direct and indirect blocks associated with the inode. If the actual number of blocks does not match the computed number of blocks, `fsck` warns of a possible file-size error. This is only a warning because HP-UX does not fill in blocks in sparse data files.

A directory inode within the HP-UX file system has the mode word set to `directory`. The directory size of a file system using 14-character filename limits must be a multiple of 32 characters, because a directory entry contains 32 bytes of information. The number of blocks actually used for the directory should match that indicated by the inode size. `fsck` reports any directory misalignment, but cannot correct it.

Block Count

`fsck` checks the block count of two types of data blocks:

- Ordinary data blocks containing information stored in a file. `fsck` does not attempt to check the validity of the contents of an ordinary data block.
- Directory data blocks containing directory entries.

Indirect blocks are owned by an inode; thus, inconsistencies in indirect blocks affect the inode that points to the block. `fsck` checks indirect blocks for the following block-count inconsistencies:

- Blocks already claimed by another inode.
- Block numbers outside the range of the file system.

`fsck` detects and corrects the indirect-block inconsistencies iteratively, by the same scheme used for direct blocks.

`fsck` checks each directory data block for inconsistencies involving:

- Directory inode numbers pointing to unallocated inodes. If a directory entry inode number points to an unallocated inode, `fsck` removes the directory entry.
- Directory inode numbers larger than the number of inodes in the file system. If a directory entry inode number points beyond the end of the inode list, `fsck` removes the directory entry. This occurs if bad data is written into a directory data block.
- Incorrect directory inode numbers for "." and ".." (current and parent directories, respectively).

  The directory inode number entry for "." should be the first entry in the directory data block. Its value should be equal to the inode number for the directory data block.

  The directory inode number entry for ".." should be the second entry in the directory data block. Its value should be equal to the inode number for the parent of the directory entry (or the inode number of the directory data block if the directory is the root directory).

  If the directory inode numbers for "." and ".." are incorrect, `fsck` replaces them with correct values.

File-System Connectivity

`fsck` checks the general connectivity of the file system. If it finds directories not linked into the file system, `fsck` links the directory back into the file system by placing them in the `/lost+found` directory.

Uncorrectable File System Corruption

In certain instances, `fsck` may be unable to check and repair the file system (for example, if all copies of the superblock are lost). The `fsdb` (file system debugger) command is provided for such situations.

Caution `fsdb` should be used only by an HP-UX file system expert, since it can easily destroy the entire file system. Refer to the *fsdb*(1M) entry in the *HP-UX Reference* for details.

Transferring Files between HP-UX and Other Systems

Not all computers use the HFS File System. Among HP products, for example, the Series 200 HP-UX 2.x used the Bell System III file system (BFS), and the Series 500 HP-UX system used Structured Directory Format (SDF). To accommodate variances, HP-UX supports several utilities and services for transferring files, including to other vendors' operating systems.

Table 8-6 shows what to use when transferring information between HP-UX and various systems. In some cases (such as with networking products), optional products must be present.

Table 8-6. Utilities and Services for File Transfer

Utility/Service	When to Use	For More Information ...
`kermit`	When both systems, connected by serial lines, run `kermit`. `kermit` transfers data between HP-UX and many incompatible operating systems.	*Kermit* chapter of *Remote Access: User's Guide*
LIF Utilities	When transferring files between HP-UX and systems that support the LIF file format, including HP-UX Basic, Pascal, and other HP-UX systems.	*lif*(4) in the *HP-UX Reference*
`uucp`	When the other system is a UNIX system (including HP-UX), connected by modem lines, direct connection, or X.25 network, and with UUCP utilities installed. `uucp` automatically reconciles differences in file format between systems.	*UUCP* chapter of *Remote Access: User's Guide*
ARPA/Berkeley Networking Services	When the both systems are connected via LAN to the same services. The other system can be an HP-UX or UNIX system, or an MS-DOS personal computer. ARPA/Berkeley Networking Services reconcile automatically any differences in file format between systems.	■ *Using ARPA Services* ■ *Installing and Administering ARPA Services* ■ *ftp*(1M) in the *HP-UX Reference*
HP FTAM/9000	When both systems are networking via OSI and using FTAM (OSI File Transfer, Access, and Management). OSI is a multi-vendor standard compatible with UNIX and non-UNIX operating systems. FTAM handles binary and ASCII file transfers, but does no data conversion.	■ *HP FTAM/9000 User's Guide* ■ *Installing and Administering HP FTAM/9000*
Network Services/9000	When transferring files over LAN to any HP-UX platform.	*Network Services/9000 User's Guide*

Table 8-6. Utilities and Services for File Transfer (continued)

Utility/Service	When to Use	For More Information ...
NFS	When users on different HP-UX (and other UNIX) systems want to share files. Explicit file transfers are unnecessary because file-system access is transparent.	■ *Using NFS Services* ■ *Installing and Administering NFS Services*
BIF Utilities	When transferring files between Series 300/400 systems and the HP Integral Personal Computer or Series 200 computers.	*bif*(4) in the *HP-UX Reference*
SDF Utilities	When transferring files between Series 300/400/700/800 and Series 500 computers.	*sdf*(4) in the *HP-UX Reference*
`cpio`	When transferring files by magnetic tape (cartridge or reel-to-reel) to another UNIX system that supports the `cpio` format. *NOTE:* `cpio` can be used with the `tcio` command to ensure smoother tape access.	*cpio*(1) in the *HP-UX Reference*
`ftio`	When copying files to magnetic tape (cartridge, reel-to-reel, or DDS format). Faster throughput than either `tar` or `cpio`.	*ftio*(1) in the *HP-UX Reference*
`tar`	When transferring files by magnetic tape (cartridge, reel-to-reel, DDS format) to another UNIX system supporting the `tar` format. *NOTE:* `tar` can be used with the `tcio` command to ensure smoother tape access. However, `tcio` only works on cartridge tapes.	*tar*(1) in the *HP-UX Reference*

8

Table 8-6. Utilities and Services for File Transfer (continued)

Utility/Service	When to Use	For More Information ...
`tcio`	When transferring files between cartridge tape units (including autochanger) and a controlling HP-UX computer. `tcio` is typically used with `cpio` or `tar`.	*tcio*(1) in the *HP-UX Reference*
`fbackbup`, `frecover`	When transferring (typically backing up and restoring) files to magnetic tape, standard out, DAT tape, rewritable magneto-optical disk, or to a file. Combines features of `dump` and `ftio`.	*fbackup*(1M) and*frecover*(1M) in the *HP-UX Reference*

File Protection

When created, each file in the file system is assigned a set of file protections stored in the file permissions bits (often called the file's mode). The file permission bits determine which classes of users may read from the file, write to the file, or execute the program stored in the file. Read, write, and execute permissions for a file can be set for the file's owner, all members of the file's group (other than the file's owner), and all other system users.

These three classes of users (user, group, and other) are mutually independent; that is, no member of one class of users is included in any other class of users. When a file is created, it is associated with an owner and a group ID. For example, a file created by `pjw` in group `dbase` is listed as being owned by user `pjw` of group `dbase`. These values specify which user owns the file and which group has special access capability.

The default permissions of a file are initially determined by `umask` (set systemwide, in the users' environment file, or on the command line), or by parameters passed to `creat`, `mknod`, or `mkdir` system calls when the file is created. The permissions can be changed with the `chmod` command.

File permissions are represented as the binary form of four octal digits as shown in Figure 8-10. The initial discussion deals with only the three least significant digits. When the most significant digit is not specified, its value is assumed to be zero (0).

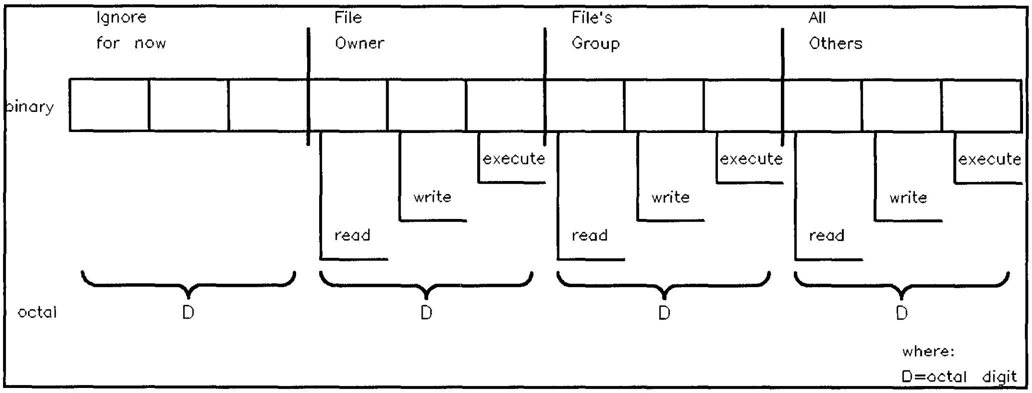


Figure 8-10. File Permission Bits

Each octal digit represents a three-bit binary value: one bit specifies read permission, one bit specifies write permission, and one bit specifies execute permission. If the bit value is one, then permission is granted for the associated operation. Similarly, if the bit value is zero, permission is denied for the associated operation.

For example, assume a file's permission bits are set to 754 (octal). Octal 754 is equivalent to 111 101 100 binary. The `ll` command represents this as `rwxr-xr--`. As shown in Figure 8-11, the file's owner may read, write, and execute the file, while read and execute permission is granted to members of the file-owner's group. This includes any user (except the file's owner) whose effective group ID equals the ID of the file's group, or whose group access list includes the file's group ID. All other system users may only read the file.

Note, if a file has associated Access Control List (ACL) entries, a **+** is displayed following the permissions. By default, the **chmod** command deletes any ACL entries, but you can use the **-A** option to preserve them. For more information on ACLs, refer to *A Beginner's Guide to HP-UX*, *HP-UX System Security*, and *acl*(5) in the *HP-UX Reference*.

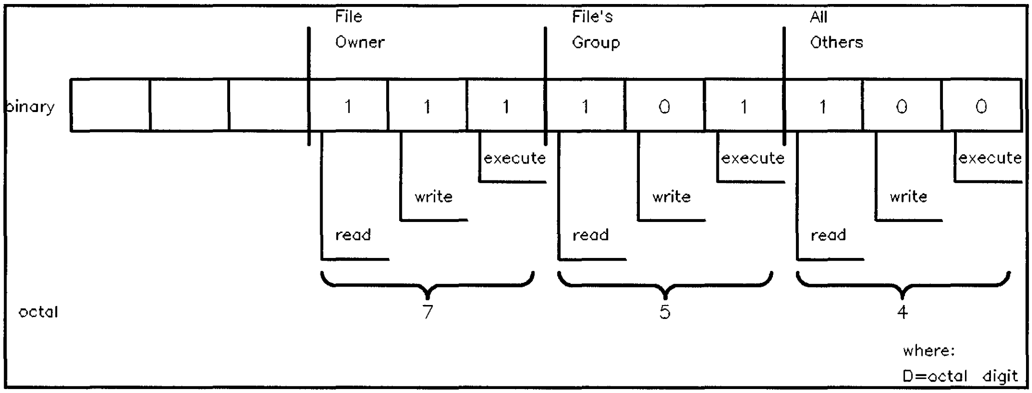


Figure 8-11. File Permission Bits of `rwxr-xr--`

Protecting Directories

Directories, like all files in the HP-UX file system, have permissions. The format of a directory's permission bits is identical to that of an ordinary file; however, the read, write, and execute permissions have a slightly different meaning when applied to a directory.

- Read permission grants access to display the contents of a directory.
- Write permission grants access to add a file to the directory, rename a file within the directory, and remove a file from the directory. Users (even superusers) cannot write directly to the directory itself. Only the kernel can write directly to directories.
- Execute permission grants access to search a directory for a file. If execute permission is not set, the files below that directory in the file-system hierarchy cannot be accessed, even when you supply the file's correct path name.

8

Setting the sticky bit on a directory provides additional protection to files within the directory: files cannot be removed from the directory except by the owner of the file, the owner of the directory, or a user having appropriate privileges. (See *rm*(1) in the *HP-UX Reference Manual*.)

Setting Effective User and Group ID Bits (suid, sgid)

As discussed in Chapter 5, "Process Management," a process has effective user and group IDs that can be used to ensure file security. Using user and group IDs, a file can be protected so that when executed, the process's effective IDs are identical to the file owner's IDs. This capability is specified through the most significant digit of the four octal file protection digits. The most significant digit is represented by a three-bit binary value. When its most significant bit is 1, the effective user ID of the process executing the file is set equal to the user ID of the file's owner. This bit is called the **set user ID bit** (`suid` or `setuid`). Similarly, if the middle bit of the most significant octal digit is 1, then the effective group ID of the process executing the file is set equal to the group ID of the file's group. This bit is called the **set group ID bit** (`sgid` or `setgid`).

If the `sgid` bit is set for an ordinary file, and the file does not have group execute permission, the file is in enforcement locking mode. Refer to the section "File Sharing and Locking" later in this chapter, or to the *lockf*(2) entry in the *HP-UX Reference*.

If the `suid` bit is set for a directory, the directory is **hidden**. A hidden directory is part of the context-dependent file (CDF) structure used when running an HP-UX cluster. For more information on CDFs, refer to *Managing Clusters of HP 9000 Computers: Sharing the HP-UX File System*.

For example, suppose the file's permission bits are 6754. The binary equivalent of octal 6754 is 110 111 101 100. These permissions are illustrated in Figure 8-12 and explained in Table 8-7.

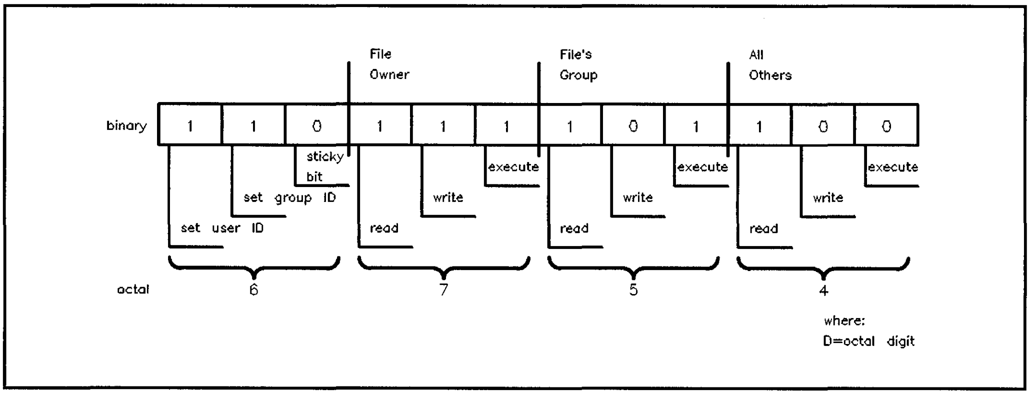


Figure 8-12. Permission Bits of an suid/sgid file set to `rwsr-sr--`

8

Table 8-7. Explanation of File Permission Bits rwsr-sr—

Octal Digit	Binary Form	Permission	Meaning
6	1	set user ID	Effective user ID of the process executing this file is set equal to the real user ID of the file's owner.
	1	set group ID	Effective group ID of the process executing this file is set equal to the group ID of the file's group.
	0	sticky bit	The sticky bit is not set; see "Protecting Directories".
7	1	read	File owner may read the file.
	1	write	File owner may write to the file.
	1	execute	File owner may execute the file.
5	1	read	Members of the file's group (other than the file's owner) may read the file.
	0	write	Members of the file's group (other than the file's owner) cannot write to the file.
	1	execute	Members of the file's group (other than the file's owner) may execute the file.
4	1	read	Any other user may read the contents of the file.
	0	write	Other users cannot write to the file.
	0	execute	Other users cannot execute the program contained in the file.

Access Control Lists

Access control lists (ACLs) offer a finer degree of file protection than traditional file-mode protection bits. With ACLs, you can allow or restrict file access to individual users, regardless of what group the users belong. For additional information see *acl*(5) in the *HP-UX Reference Manual* manual and/or *HP-UX System Security*, HP part number B1862-90009.

File Sharing and Locking

In a multi-user, multi-tasking environment such as HP-UX, it is often desirable to control interaction with files. Many applications share disk files, and the status of information contained in them could have serious implications to the user (such as lost or inaccurate information).

Imagine we are responsible for maintaining on-line technical reports for a myriad of projects, and we have many different people who must have simultaneous access to these reports. The content of a given report at a given time could significantly affect a company decision, and so we want a way to control how records are accessed.

One potential problem could arise if one person (let's call him George) adds to or modifies information in a report while someone else (Sarah) is working on it. Sarah is unaware of changes that George has just made in the report. And once she is done, Sarah overwrites the information George added. The result is that we have lost *all* of George's information, and when Sarah added data she was unaware of information that might have been pertinent.

Advisory Locks

A solution to this problem common to file sharing is called **file locking**. In HP-UX, file locking is done with the `lockf` or `fcntl` system calls, which handle two modes of functionality. **Advisory lock**s are placed on disk resources to inform (warn) other processes desiring access that a file is currently being accessed or modified. Advisory locks are only valuable for cooperating processes that are both aware of and use file locking.

8

In our example, the programs used to access the on-line reports can use advisory locks. When George begins to work on the Marketing project his program can call `lockf` and set an advisory lock. A few minutes later when Sarah tries to access records in the Marketing report, she would get an error message indicating that the report is busy. Her program could wait until George is done and then access the report, by using the system call, `lockf`.

Enforcement Mode

Even if we use advisory locks in our example, Sarah would still be able to overwrite the Marketing report if she uses commands or utilities that do not check for advisory locks. She needs some way to ensure that no records are written until George finishes accessing the report. HP-UX does this with **enforcement mode**. When a process attempts to read or write to a locked record in a file opened in enforcement mode, the process sleeps until the record is unlocked. Enforcement mode can be used only on regular files.

Enforcement mode is enabled when the set-group-id bit (`sgid`) is set but the group execute bit is not set. For example, if we opened a file which normally has its file permission bits set to 644, a long listing of the file would look something like:

```
-rw-r--r--   1 george   fiscal      512 May  7 16:11 Marketing
```

Enforcement mode can be enabled by typing:

```
chmod g+s Marketing
```

This command turns on the `sgid` bit, resulting in file protection of 2644. Enforcement mode can also be enabled by using the `chmod` system call. After enforcement mode is enabled, a long listing shows:

```
-rw-r-Sr--   1 george   fiscal      512 May  7 16:11 Marketing
```

Using enforcement mode, George can prevent Sarah from overwriting his changes, and Sarah would have the data that George has added.

When attempting to access a file locked under enforcement mode, the process sleeps until the file is released. This provides a means for one process to control execution of another. Be careful when doing this, because a system deadlock is possible.

Locking Activities

All file locking is controlled with the `lockf` or `fcntl` system calls. `lockf` controls four file actions:

- Testing file accessibility by checking to see if another process is present on a specific file record.
- Attempting to lock a file. If the record is already locked by another process, `lockf` puts the requesting process to sleep until the record is free again.
- Testing file accessibility, locking the record if it is free, and returning immediately if it is not.
- Unlocking a record previously locked by the requesting process.

When the locking process either closes the locked file or terminates, all locks placed by that process are removed. For more details on how specific locking activities work on HP-UX, refer to the *lockf*(2) and *fcntl*(2) sections of the *HP-UX Reference* manual.

9

Logical Volume Manager

The **Logical Volume Manager (LVM)** is a disk management subsystem (Series 800 only) that lets you allocate disk space according to the specific or projected sizes of your file systems or raw data. LVM logical volumes provide other capabilities that are unavailable when using fixed partitions:

- LVM file systems can exceed the size of a physical disk. This feature is known as **disk spanning**, because a single file system can span disks.
- Up to three copies of identical data can be stored and updated simultaneously using LVM. This feature is known as **mirroring**, and requires an optional product, HP MirrorDisk/UX (part number B2491A).

Note

If you have a pre-installed 9.0 system, LVM is ready to use.

If you are upgrading to 9.0, you must "migrate" your disks to LVM: You must backup all data from disks, set up logical volumes on disks, and restore data to disks.

Migrating the root disk requires re-installation; see *Installing and Updating HP-UX*.

LVM is most useful for:

- managing file systems whose size is likely to grow. With LVM, you can specify the size of a file system based on need, expand a file system by adding disk space from anywhere in your system.
- large-scale applications, such as databases and CAD/CAE systems, whose data requirements often exceed the capacity of a single disk.
- disk mirroring for increased data integrity and high availability. LVM's mirroring capabilities enable you to perform some system administration tasks, such as backups, without bringing the system down.

The LVM Paradigm

Implementing LVM requires a new way of thinking about disks and file system configurations.

- In a traditional HP-UX configuration, you consider your disks individually and in terms of their variously sized sections (or partitions), each of which holds a file system, swap space, boot area, or raw data.
- With LVM, you do not use disk sections. Instead, you consider the disks as a pool (or volume) of data storage, consisting of equally sized (4 MB) **extents**, multiples of which are allocated for file system, swap, and other purposes. HP LVM lets you specify size in megabytes.

Setting up LVM requires you to follow a procedure to reorganize your system. This is documented in *Installing and Updating HP-UX* and *System Administration Tasks* manual.

An LVM system consists of groupings of disks initialized for LVM and organized into **volume groups**. A volume group might consist of one or many LVM disks; your entire system might consist of one or several volume groups.

Just as volume groups are groupings of one or more LVM disks, volume groups are also subdivided into virtual disks, called **logical volumes**. Logical volumes can

- encompass space on one or more LVM disk
- span LVM disks
- represent only a portion of an LVM disk.

You apportion disk space in a volume group by creating logical volumes. The size of a logical volume is determined by its number of extents, each being 4 MB by default and is configurable. You then assign file systems, swap, and dump to logical volumes.

In Figure 9-1, logical volume `/dev/vg00/lvol1` might contain a file system, `/dev/vg00/lvol2` might contain swap space, and `/dev/vg00/lvol3` might contain raw data.

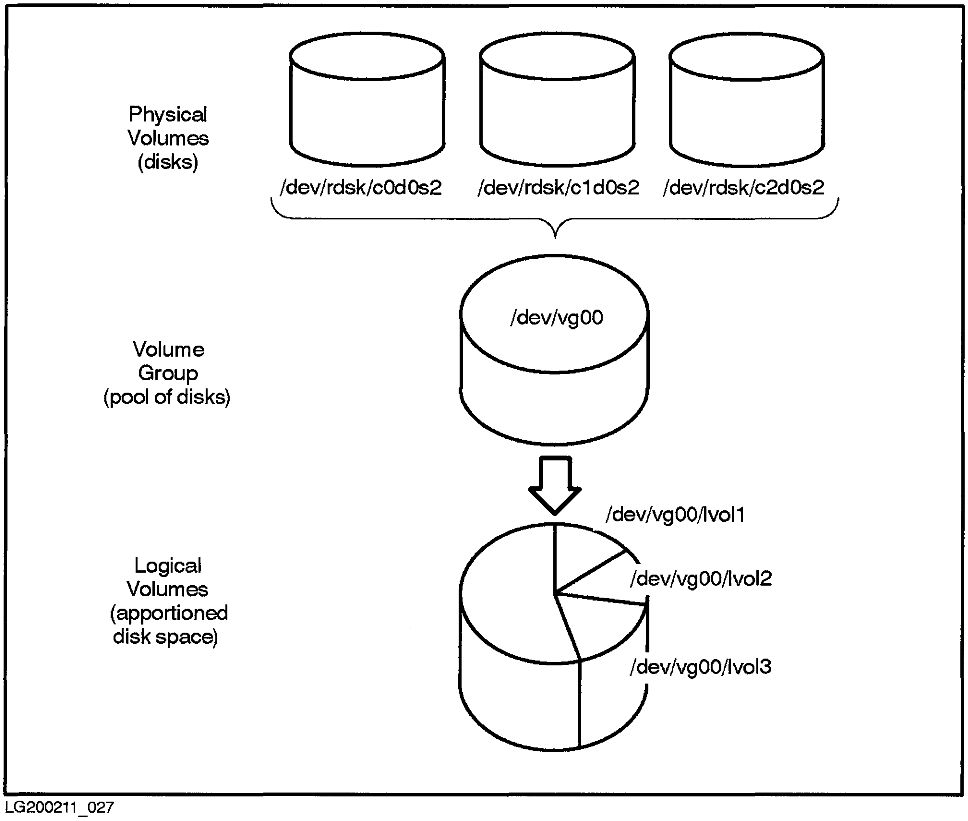


Figure 9-1. Disk Space Apportioned into Logical Volumes

You can use SAM to create a file system in a logical volume of a specified size, and then mount the file system.

Or using commands, you can create and then extend a logical volume to allocate sufficient space for file system, user application, or raw data. You would then create and mount new file systems or install your application in the logical volume. You use the same approach when increasing the capacity of a file system created on a logical volume.

All procedures for implementing LVM are detailed in Chapter 7 of the *System Administration Tasks* manual. All LVM commands are documented in the *HP-UX Reference Manual.*

Understanding Migration from Sections to Logical Volumes

Before you can use LVM on a system whose file systems exist in fixed disk sections, you must first migrate to LVM. You can migrate your entire system (including the root disk) or only the more dynamic file systems or sections of raw data.

Migration is documented in *Installing and Updating HP-UX*, but briefly, the procedure involves the following:

- Plan your new configuration to take advantage of LVM features. Use the tool, `lvmmigrate` (available on the system tape) to plan your allocation of space in logical volumes.
- Copy all customized files to a safe directory.
- Back up the contents of your disks. You *must* do this because LVM applies new data structures to your disks and anything on your disks will be overwritten. You can use `fbackup` or `cpio` to back up your file systems, and `dd` to back up raw data.
- Install HP-UX on the root disk, which allows you to create new logical volumes for root and swap.
- Restore files and raw data, using `frecover` for file systems and `dd` for raw data.
- Return customized files to the root directory.
- Use SAM to complete the migration of remaining data to logical volumes. This involves creating and extending logical volumes to hold your data, then recovering backed-up data onto the logical volumes.

Note LVM data structures consume a small, but significant amount of overhead space. Therefore, be sure to plan thoroughly before implementing LVM.

LVM Terminology

The following LVM terminology is new to HP-UX:

allocation policy
: The LVM allocation policy governing how disk space is distributed to logical volumes and how extents are laid out on an LVM disk.

  LVM allocates disk space in terms of strict vs. non-strict and contiguous vs. non-contiguous. Strict allocation requires that mirror copies reside on different LVM disks. Contiguous allocation requires that no gaps exist between physical extents on the disk.

bad block relocation
: An LVM feature for recovering corrupted blocks of data from HP-FL and SCSI disks, by redirecting the data to another block of media on the disk. Bad block relocation uses a remainder of space in the disk layout. This feature can be enabled or disabled with the *lvchange*(1M) command.

  This feature is not supported on HP-IB disks, nor is it supported for the root logical volume.

block
: The smallest unit of space transferrable to and from a disk. LVM transfers data in blocks of `DEV_BSIZE` (1024) bytes.

extent
: Fixed-size addressable areas of space on an LVM disk or in memory. On disk, these contiguous areas are called physical extents, and are by default 4 MB. Physical extents map to areas in memory, called logical extents. (For more information on extents, see "Allocating Disk Space for Logical Volumes" later in this chapter.)

I/O channel separation
: A configuration of disks useful for segregating highly I/O-intensive areas. For example, you might have a database on one channel and file systems on another.

  When mirroring logical volumes using HP MirrorDisk/UX, you can spread the mirrored copies over different I/O channels to increase system availability.

logical volume A storage device, much like a disk section but of flexible size, that can hold a file system (including root), raw data, application program, or swap. Logical volumes can be mirrored using an optional product, HP MirrorDisk/UX.

Because its data is distributed *logically* (rather than physically), a single logical volume can be mapped to one LVM disk or span multiple disks, but the logical volume itself is used as a single virtual disk.

The `lvdisplay` command can be used to verify distribution of logical volumes on disks.

Logical Volume Manager (LVM) An operating system software module that implements virtual (logical) disks to extend, mirror, and improve the performance of physical disk access.

LVM disk A disk that has been initialized for LVM. (Also see "physical volume.")

mirroring Simultaneous replication of data (up to three copies) using an optional product, HP MirrorDisk/UX (part number B2491A). (HP MirrorDisk/UX is not supported on HP-IB disks.)

Duplicated information storage ensures a greater degree of data availability. Using HP MirrorDisk/UX, LVM maps logical volumes to multiple LVM disks, thus providing the means to recover easily from the loss of one copy (or two copies in the case of three-way mirroring) of data.

Mirroring can provide faster access to data for database applications using more data reads than writes.

physical volume A disk that has been initialized by LVM for use in a volume group; also called an LVM disk.

As with standard disks, an LVM disk (physical volume) is accessed via a raw device file for section two (for example, `/dev/rdsk/c3d0s2`).

You can use SAM or the `pvcreate` command to initialize a disk as a physical volume (LVM disk).

physical volume group
A configuration of physical volumes (LVM disks) within a volume group into a physical volume group, on the basis of the I/O channel or interface adapter to which they are connected to achieve higher availability of mirrored data. (Also see "I/O Channel Separation", later in this chapter.)

quorum
The requirement that a volume group have at least half the configured LVM disks present to change or activate that volume group. If there is no quorum, LVM prevents the change.

Quorum is checked both during configuration changes (for example, when creating a logical volume) and at state changes (for example, if a disk fails). If quorum is not maintained, LVM will not acknowledge the change.

Quorum ensures the consistency and integrity of the volume groups. The `vgchange` command with the `-q n` option can be used to override quorum check.

root logical volume
The logical volume containing the HP-UX kernel.

synchronization
The process of updating "stale" (non-current) copies of mirrored logical extents by copying data from a "fresh" (current) copy of the logical volume. Synchronization keeps mirrored logical volumes consistent by ensuring that all copies contain the same data.

volume group
A collection of one or more LVM disks from which disk space may be allocated to logical volumes. A disk can belong to only one volume group. A volume group is accessed through the `group` file (for example, `/dev/vg01/group`) in that volume group's directory.

You can use SAM or the `vgcreate` command to create a volume group.

HP-IB devices may not be combined with devices of any other interface in a volume group.

Using LVM Device Files

All LVM components are represented by device files located in the `/dev` directory. (You might think of device files as agents for managing the interactions with the disk space.) The LVM device files are created by both SAM and HP-UX commands and by default follow a standard naming convention. However, you can choose more intuitive names for volume groups and logical volumes.

Table 9-1. LVM Device File Names (defaults)

LVM Component	Block Special File	Raw Special File
LVM disk[1]	`/dev/dsk/c`*n*`d0s2`	`/dev/r dsk/c`*n*`d0s2`[2]
Volume Group	`/dev/vg`*n*[3]	N/A
Logical Volume	`/dev/vg`*n*`/lvol`*n*[4]	`/dev/vg`*n*`/r lvol`*n*[2]

1 LVM disks (physical volumes) are named for section two, the entire disk.

2 The `r` in the file name denotes "raw" (also called "character") and refers to the way data is streamed, in characters rather than blocks, or as raw data rather than blocks of data.

3 (This is not a block special file, but a directory.) Directories for volume groups are numbered `vg00`, `vg01`, `vg02`, ... `vg`*nn*, in order created.

4 Device files for logical volumes are listed in the subdirectory for the volume group to which they belong.

Table 9-2 shows which type of device file to use for common LVM tasks.

Table 9-2. When to Use Block vs. Raw Special Files for LVM

LVM Task	Special File Type	
	Block	Raw
Creating a physical volume (SAM or `pvcreate`)		X
Adding a physical volume to a volume group	X	
Creating a file system in a logical volume (SAM or `newfs`)		X
Adding or extending disk space to a logical volume (SAM or `extendfs`)		X
Mounting or unmounting a file system	X	
Accessing raw data in a logical volume		X

Major and Minor Numbers of LVM Data Structures

The LVM device drivers find data via the major and minor numbers of the file system data structures. The device file for an LVM disk is shown in the long listing in Figure 9-2:

```
% ll /dev/dsk/c3d0s2
brw-r----- 1 root sys   7 0x000302 Dec 20 15:38 /dev/dsk/c3d0s2
```

Figure 9-2. Device File for an LVM Disk (physical volume)

The 7 represents the major number (index to the device driver), 3 is the LU number (physical device), and 2 is analogous to the section number of a Series 800 disk.

Now compare the minor numbers (shaded) for a volume group in Figure 9-3

```
% ll /dev/vg3
crw-rw-rw- 1 root  other  64 0x030000 Jan 23 14:35 group
brw-r----- 1 root  other  64 0x030001 Jan 24 17:02 lvol1
crw-r----- 1 root  other  64 0x030001 Jan 24 17:02 rlvol1
brw-r----- 1 root  other  64 0x030002 Feb  3 11:53 lvol2
crw-r----- 1 root  other  64 0x030002 Feb  3 11:53 rlvol2
brw-r----- 1 root  other  64 0x030003 Mar  6 12:01 sales
crw-r----- 1 root  other  64 0x030003 Mar  6 12:01 rsales
```

Figure 9-3. Volume Group, Showing Minor Numbers (shaded)

with the overall hexadecimal format by which they were created, in Figure 9-4:

```
31             23        15       7        0
+------------+---------+--------+---------+
|major number|VG number|reserved|LV number|
+------------+---------+--------+---------+
             |<-------minor number------->|
```

Figure 9-4. Hexadecimal format of LVM Major and Minor Numbers

The logical volume (`LV`) number is encoded into bits 0-7; the volume group number (`03`), comparable to the LU number of a non-LVM device file but in a different position, is encoded into bits 16-23. The major number is encoded into bits 24-31. Note too, the block and raw special files created for the logical volume `/dev/vg00/sales`.

Within each volume group directory is a special file, **`group`**, with the volume group major number `64`, logical volume number `0`, and volume group number. Volume group numbering begins with zero (`vg00`), while logical volumes begin with one (`lvol1`). This is because the logical volume number corresponds to the minor number and the volume group's **`group`** file is assigned minor number `0`.

HP-UX accommodates up to 256 volume groups and 255 logical volumes per volume group.

Understanding Volume Groups

If you install LVM on your root file system, you have `/dev/vg00`, your first LVM volume group. (If you have not installed LVM on your root file system, you must use SAM or manually create `/dev/vg00`.) You can then apportion disk space from the volume group into logical volumes.

As we have seen, volume groups are visible in HP-UX as device files in the `/dev` directory. The directory includes device files for the logical volumes belonging to the volume group. The volume group directory also contains a `group` file, which provides LVM data structures with information about the entire volume group. (These data structures are described in "Characteristics and Layout of LVM Disks", later in this chapter.)

To see the contents of a volume group, run the `vgdisplay` command. See *vgdisplay*(1M) in the *HP-UX Reference Manual* for full explanation of the command and its options. (For a very graphic representation of a sample LVM configuration, showing volume groups organized functionally, see the LVM chapter of *Solving HP-UX Problems*.)

Figure 9-5 shows verbose output of `vgdisplay` on a system comprised of one volume group, one LVM disk (physical volume), and two logical volumes. You can also see that volume group `/dev/vg00` consists of one physical volume (`PV`) out of a maximum 32, two logical volumes (`LV`) out of a maximum 255, and that each physical extent (`PE`) is four megabytes. Of a total 632 physical extents available, 170 have been allocated, leaving 462 free. Status information reveals that the logical volume contents are synchronized, or up-to-date.

HP-IB Limitations

HP-IB disks must be used in their own volume group and cannot be combined in volume groups with HP-FL or SCSI disks. This is because HP-IB disks can handle only limited LVM capabilities: HP-IB does not support bad block relocation or disk mirroring.

```
% vgdisplay -v
 --- Volume groups ---
 VG Name                     /dev/vg00
 VG Status                   available
 Max LV                      255
 Cur LV                      2
 Open LV                     2
 Max PV                      32
 Cur PV                      1
 Act PV                      1
 PE Size (MB)                4
 Max PE per PV               1016
 VGDA                        2
 Total PE                    632
 Alloc PE                    170
 Free PE                     462
 Total PVG                   0
    --- Logical volumes ---
  LV Name                    /dev/vg00/lvol1
  LV Status                  available/syncd
  Current LE                 10
  Allocated PE               10
  Used PV                    1

  LV Name                    /dev/vg00/lvol2
  LV Status                  available/syncd
  Current LE                 20
  Allocated PE               20
  Used PV                    1
  --- Physical volumes ---
  PV Name                    /dev/dsk/c2d0s2
  PV Status                  available
  Total PE                   632
  Free PE                    462
```

Figure 9-5. Sample Verbose Output of `vgdisplay`

Consider newfs Limitations when Organizing Volume Groups

Best system performance is achieved if you organize your volume groups by identical disk type. This means that if you create a file system that spans LVM disks, be sure that the logical volume in which the file system resides spans identical disk types.

This guideline derives from the behavior of *newfs*(1M): when you run the `newfs` command to construct a file system, you can specify only one disk type, which `newfs` then uses to lay out the disk. If more than one disk type is used for the file system, the logical volumes on the specified disk type will perform well, but the unspecified disk type will undoubtedly be less efficient. Thus, try to maintain consistency of disk type among physical volumes used for any logical volume that holds a file system.

You might already know the disk type to which your file system is being mounted. However, on a system set up with LVM, the `lvdisplay -v` command is the most accurate starting point for finding the correct disk type.

Another consideration when grouping disk drives is their relative size: The size of disk drives within a volume group should be similar (within a factor of two between largest and smallest) to avoid wasting space on LVM data structures.

Consult /etc/diskinfo for Disk Attributes

When setting up logical volumes, you might find the information presented by `/etc/diskinfo` helpful in determining how to allocate your file systems.

The `/etc/diskinfo` command describes disks attributes such as number of bytes per sector, device type, and timing information. You can also use it for more information about a device file listed in `/etc/checklist`. The following example shows relevant output from `/etc/diskinfo` for a SCSI disk:

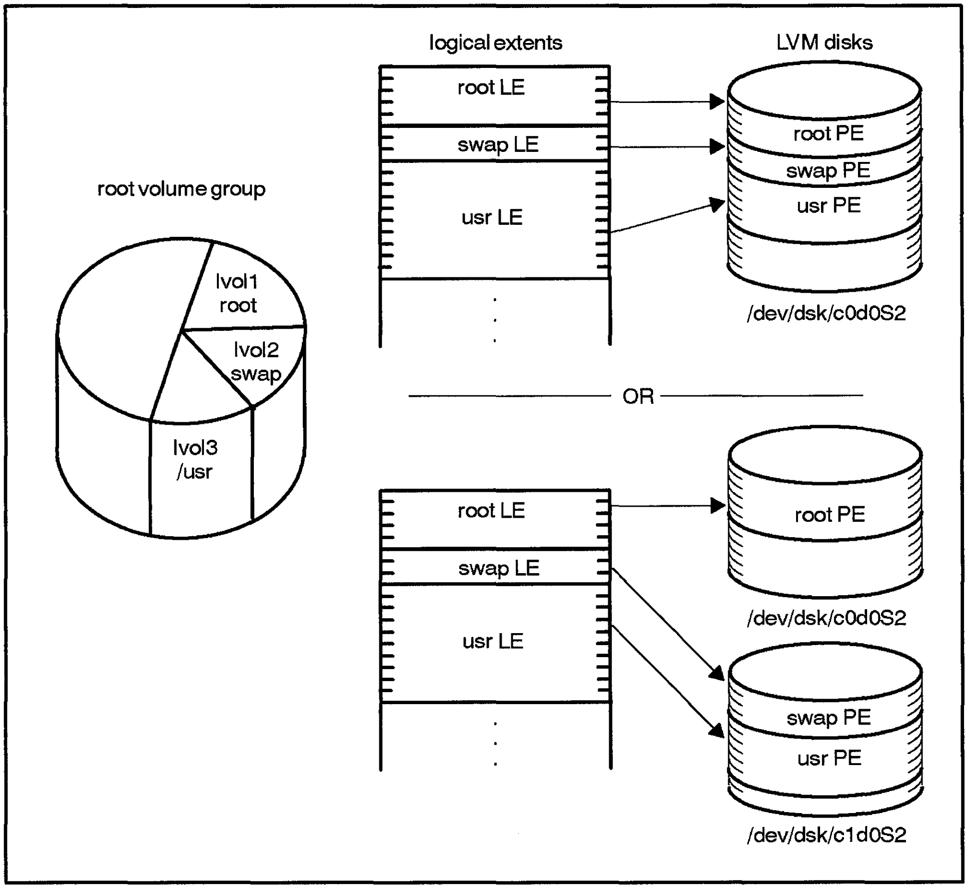
```
# /etc/diskinfo -v /dev/rdsk/c3d0s2
SCSI describe of /dev/rdsk/c3d0s2:
      vendor: HP
             product id: 2213A
      type: direct access
      size: 648192 Kbytes
      bytes per sector: 512
      blocks per disk: 1296511
```

The Root Volume Group

- In a traditional configuration, the root disk contains all the attributes required for initial booting, plus system files, dump, and primary swap—all on one disk.
- In LVM, the concept of a single root disk is replaced by a **root volume group**. The root volume group contains all the same elements as a root disk, and `/`, `swap`, and `/usr` fit on a single 300-MB disk.

  The root logical volumes can be dispersed over more than one LVM disk, as shown in the lower example in Figure 9-6. In practice, however, your root volume group might be nearly identical to a traditional root disk. By default, the HP-UX installations programs automatically fill one disk at a time.
- You can mirror root logical volumes, using an optional product, MirrorDisk/UX.

Figure 9-6 shows a root volume group whose logical extents are mapped to either one or two LVM disks.



LG200211_028

Figure 9-6. Root Volume Group

When you install the system, having decided to implement LVM, the migration and installation processes set various options ensuring that your root volume group is properly configured. Later, if you choose to establish another root volume group, you might want to refer to the guidelines found in "Guidelines for Configuring the Root Volume Group", later in this text.

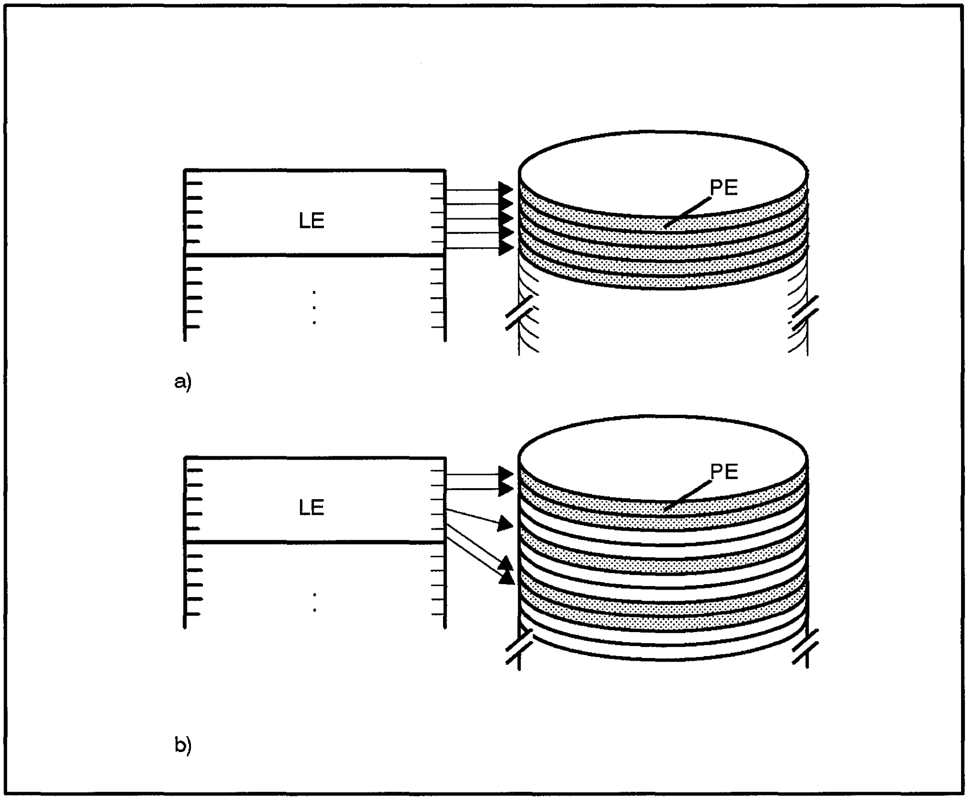
Underlying the root volume group is an LVM disk data structure different from that of non-bootable LVM disks. Bootable LVM disks have sectors reserved for a boot data reserved area (BDRA) and a LIF volume containing boot programs. For an LVM disk to be bootable, it must be initialized using the *pvcreate*(1M) command with the `-B` option (see "Managing Logical Volumes" in *System Administration Tasks* manual for the procedure). The boot data reserved area and characteristics common to both bootable and standard physical volumes are discussed later in this chapter, in "Characteristics and Layout of LVM Disks".

Contiguous vs. Non-Contiguous Allocation of LVM Disk Space

Just as traditional disk space is allocated contiguously if possible, and then by fragments, for efficient use of disk space, so LVM LVM allocates disk space with regard to contiguity.

As illustrated in Figure 9-7,

- Contiguous disk space allocation means that the logical extents (`LE`) of a logical volume must be mapped to adjacent physical extents (`PE`) on an LVM disk (example `a`)).
- Non-contiguous disk space allocation means that the logical extents of a logical volume need not be mapped to adjacent physical extents (example `b`)).



LG200211_029

Disk space used by the root volume group *must* be contiguous. That is, physical extents mapping to the logical volumes of the root file system, primary swap, and dump must be contiguous. The root volume group's contiguous extents also adhere to the following requirements:

- Physical extents must be allocated in ascending order.
- No gap may exist between physical extents.
- When mirrored, all physical extents of a mirrored copy must reside on the same LVM disk.

Contiguous allocation is less flexible than non-contiguous allocation, and therefore uses disk space less economically. Non-contiguous mapping can result in a logical volume's physical extents being dispersed onto more than one LVM disk, since logical volumes can span multiple disks.

Understanding Logical Volumes

Logical volumes are allotments of disk space within a volume group. Like disk sections or partitions, logical volumes hold file systems, swap, or raw data; unlike disk sections, you can set the capacity of logical volumes according to your need.

Logical volumes consist of logical extents, which map to the physical extents of LVM disks. This mapping gives logical volumes great flexibility:

- to span more than one LVM disk.
- to be larger than a single LVM disk
- to be reduced or expanded to meet changing file system needs. (The file system must be unmounted to reduce or expand a logical volume.)
- to be mirrored, using an optional product, HP MirrorDisk/UX.

Each logical volume has a block special device file and raw special device file. These device files are listed in the subdirectory of the volume group to which the logical volumes belong.

To see a logical volume, type the **lvdisplay** command, giving as the argument the pathname of the logical volume:

```
% lvdisplay /dev/vg00/lvol1
--- Logical volumes ---
LV Name                 /dev/vg00/lvol1
VG Name                 /dev/vg00
LV Permission           read/write
LV Status               available/syncd
Mirror copies           0
Consistency Recovery    MWC
Schedule                parallel
Current LE              10
Allocated PE            10
Bad block               on
Allocation              strict
```

Figure 9-8. Sample Output of `lvdisplay`

Much of the information given—LV status, Mirror copies, Consistency Recovery, Schedule, bad block, and Allocation—pertain to mirroring capabilities provided by the optional product, HP MirrorDisk/UX. Mirroring is described later in this chapter.

Verbose output of the `lvdisplay` command shows the mapping of physical extents (PE) and logical extents (LE) of the logical volume on the LVM disk. This information can be useful when creating a file system.

```
--- Distribution of logical volume ---
PV Name            LE on PV  PE on PV
/dev/dsk/c2d0s2    10        10

--- Logical extents ---
LE   PV1                PE1  Status 1
0000 /dev/dsk/c2d0s2    0000 current
0001 /dev/dsk/c2d0s2    0001 current
0002 /dev/dsk/c2d0s2    0002 current
0003 /dev/dsk/c2d0s2    0003 current
0004 /dev/dsk/c2d0s2    0004 current
0005 /dev/dsk/c2d0s2    0005 current
0006 /dev/dsk/c2d0s2    0006 current
0007 /dev/dsk/c2d0s2    0007 current
0008 /dev/dsk/c2d0s2    0008 current
0009 /dev/dsk/c2d0s2    0009 current
```

Figure 9-9. Verbose Output of `lvdisplay`, Showing Logical Extents

Allocating Disk Space for Logical Volumes

- Using traditional HP-UX, once you partition the disk into fixed disk sections of various sizes, you make your file systems in the context of these sections. Supported section sizes are shown in `/etc/disktab`.
- Using LVM, you create a logical volume and allocate disk space for file systems, swap, or raw data in either megabytes or an LVM unit of measure called an **extent**.

By default, LVM extents are 4 MB. Extent size is configurable, provided the size chosen is a power of two (for example, 1,2,4,8); valid extents can range in size from 1 MB to 256 MB.

Note When calculating the size of a logical volume to contain a file system, base your calculations on how many megabytes a file system needs, but choose a quantity divisible by the extent size. Any quantity not divisible by extent size is rounded up.

Under most circumstances, you need never change the 4-MB default extent size. Exceptions might arise for extremely large disks. Large extents are more wasteful of disk space and smaller extent sizes allow a finer granularity of allocation. (For more information on these considerations, see "Volume Group Descriptor Area (VGDA)", later in this chapter.)

As you can see in Figure 9-9, LVM uses two kinds of extents—physical extents and logical extents. LVM stores data on disks as sets of addressable, disk blocks called physical extents. Logical volumes consist of logical extents, which map to the disks' physical extents.

An unmirrored logical volume has an identical number of physical and logical extents; a doubly mirrored logical volume has three physical extents to each logical extent.

Logical extents can be mapped non-contiguously to physical extents on LVM disks. This means that LVM can disperse data in non-consecutive physical extents on disk. An exception to this policy is the root volume group. (See "Contiguous vs. Non-Contiguous Allocation of LVM Disk Space", earlier in this chapter.)

How LVM Maps Extents to Logical Volumes

The LVM subsystem maps logical extents to physical extents via a translation table that resides on the LVM disk. When the volume group is activated, the table resides in real memory. LVM translates incoming read and write requests to the correct address of the physical extent, then sends the request to the corresponding physical block. Thus, the extent serves as a translation mechanism between the incoming request and underlying device drivers.

By default, LVM selects available physical extents from LVM disks in the order the disks were added to the volume group.

The physical-to-logical extent mapping of a logical volume can be discontiguous; a single logical volume (and therefore file system) can exist on several different LVM disks. File system performance is best, however, when its physical extents are contiguous, whether on one or multiple LVM disks.

As a system administrator, you can control to which LVM disks a logical volume is mapped (bypassing the default LVM's distribution), by using a two-step process of `lvcreate` and `lvextend`. This is documented in *System Administration Tasks* manual, Chapter 7, "Managing Logical Volumes."

LVM Disk Space Allocation Commands

The following commands let you create and view disk-space allocation:

- You can use SAM or the *vgcreate*(1M) command to set the size and number of physical extents in a volume group.
- You can use SAM or the *lvcreate*(1M) command to assign the number of physical extents in a logical volume.
- You can increase the number of physical and logical extents in a logical volume by using the *lvextend*(1M) command.
- You can display the size (in bytes) of physical extents (PE) in a volume group by using the *vgdisplay*(1M) command.
- You can display the number of logical extents (LE) in a logical volume by using the *lvdisplay*(1M) command.

Understanding Physical Volumes (LVM Disks)

Physical volumes (also called LVM disks) are disks that have been initialized (using SAM or `pvcreate`) for LVM. Whereas traditional disks are partitioned into sections, space on LVM disks is allocated for logical volumes.

Physical volumes use the same device special files as traditional HP-UX disk devices. Once a disk has been initialized, you can view its characteristics as an LVM disk by running the `pvdisplay` command, giving as an argument the disk's block special file.

```
% pvdisplay /dev/dsk/c2d0s2
--- Physical volumes ---
PV Name                    /dev/dsk/c2d0s2
VG Name                    /dev/vg00
PV Status                  available
Allocatable                yes
VGDA                       2
Cur LV                     4
PE Size (MB)               4
Total PE                   632
Free PE                    462
Allocated PE               170
Stale PE                   0
```

Figure 9-10. Sample Output of `pvdisplay`

A verbose listing (`pvdisplay -v`) of the same LVM disk shows the mapping status of all available physical extents. (In the following example, physical extents `0170` to the end are free, meaning unallocated.)

```
--- Distribution of physical volume ---
LV Name            LE of LV  PE for LV
/dev/vg00/lvol1    10        10
/dev/vg00/lvol2    20        20
/dev/vg00/lvol3    100       100
/dev/vg00/lvol4    40        40

--- Physical extents ---
PE   Status   LV                 LE
0000 current  /dev/vg00/lvol1    0000
0001 current  /dev/vg00/lvol1    0001
0002 current  /dev/vg00/lvol1    0002
0003 current  /dev/vg00/lvol1    0003
  :
0170 free                        0000
0171 free                        0000
0172 free                        0000
  :
```

Figure 9-11.
Verbose Output of `pvdisplay`, showing Mapping of Physical and Logical Extents

Sector Size of Physical Volumes (LVM Disks)

In HP-UX, different disk types have different sector sizes (shown in Figure 9-12). HP-IB and HP-FL disks have 256 bytes per sector; SCSI disks have 512 bytes per sector; multi-spindle disk arrays have 2048 bytes per sector. LVM deals with this diversity without user intervention.

The disk device drivers (such as `disc3` for SCSI) understand the language of the physical disk as well as a requirement of HP-UX, which specifies that device drivers be able to receive data in units of `DEV_BSIZE` (1024) bytes and pass the data to the disk in its own terms.

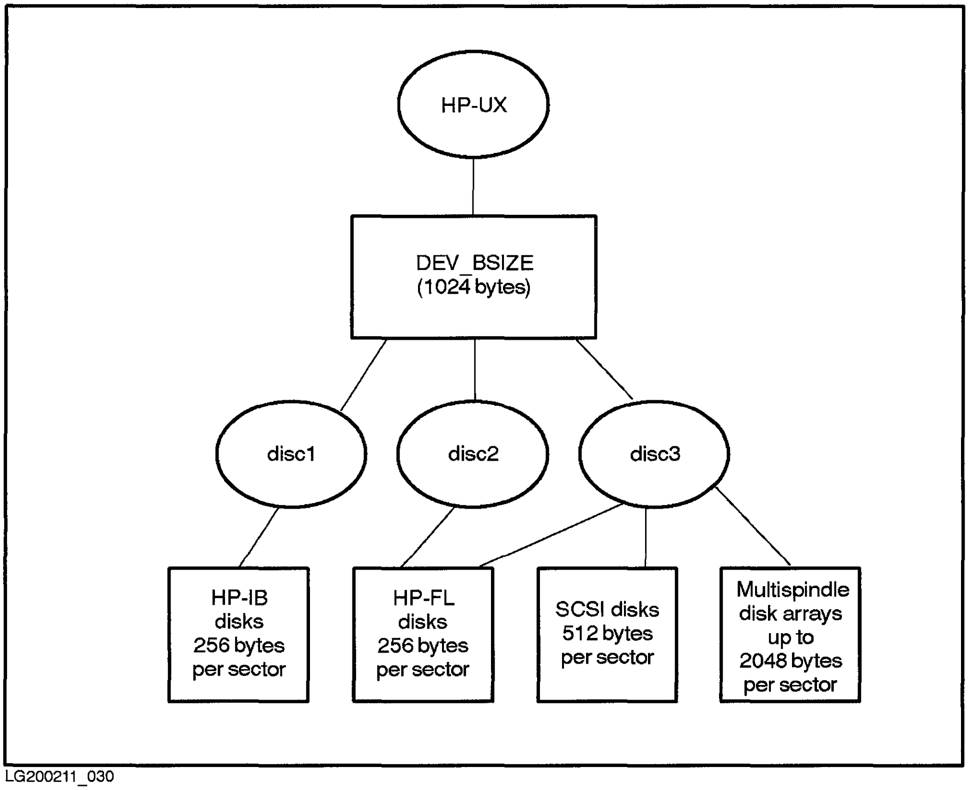


Figure 9-12. Relationship Between Disk Sector Size and DEV_BSIZE.

When using the raw interface to a device (as with a database application), data is read and written in `DEV_BSIZE` units. This can pose a performance loss for multi-spindle disk arrays, whose writes are required in units of 2 KB, the disk array's underlying sector size. For example, to transfer 1 KB (that is, one `DEV_BSIZE` unit) of data, the controlling device driver reads in 2 KB of data (even though only 1 KB is changing) in order to write in the 2-KB sector size required by the disk array. (This is called "read modify write.")

For block interface (as with a file system), there is no performance loss as long as the fragment size is 2 KB or greater.

Note Any file system that resides, even partially, on a multi-spindle disk array should have its fragment size be a multiple of 2 KB.

LVM handles all data transfers in terms of DEV_BSIZE units. In most cases, LVM communicates with the disk driver directly, to ensure that I/O block size is a multiple of DEV_BSIZE and the beginning I/O address begin on the DEV_BSIZE boundary. LVM allocates disk space in units of extents. Extent sizes range from 1 MB to 256 MB and must be a power of 2.

Characteristics and Layout of LVM Disks

There are two kinds of LVM disk layouts—for boot disks and other LVM disks—and they differ in their data structures. Non-bootable disks have two reserved areas—the physical volume reserved area (PVRA) and the volume group reserved area (VGRA), while bootable disks have additional sectors reserved for the boot data reserved area (BDRA) and LIF.

Figure 9-13 shows the layouts of bootable and non-bootable LVM disks. The text that follows describes each of the component parts and then discusses LVM overhead.

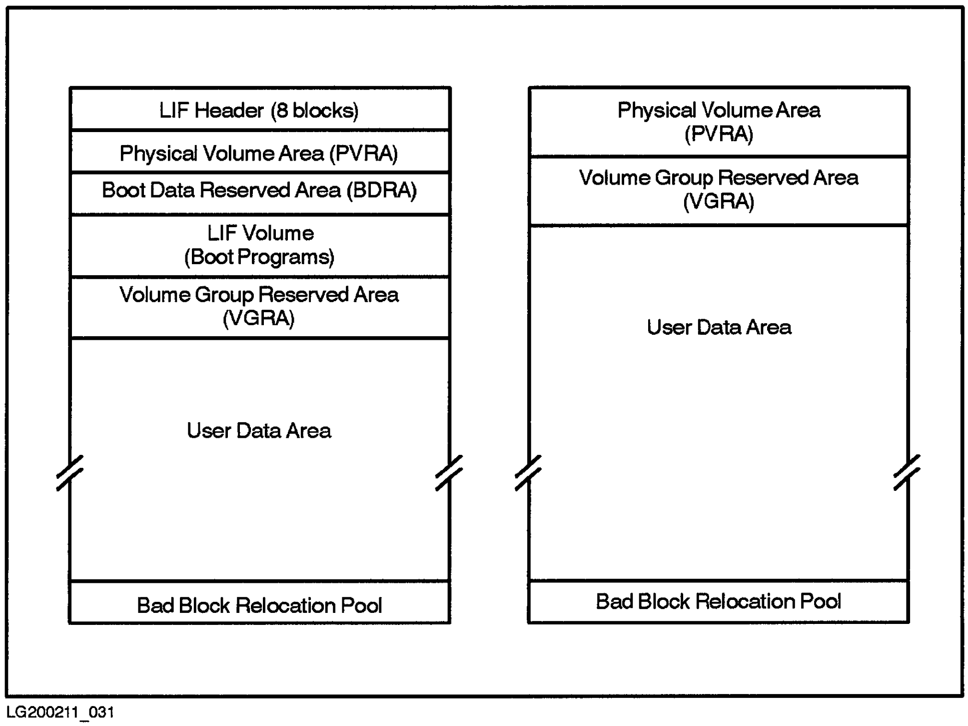


Figure 9-13.
Bootable and Non-Bootable Physical Volume Layouts, Showing Organization of Data Structures

9

Boot Data Reserved Area (BDRA)

To boot the system, the kernel activates the volume group to which the system's root logical volumes belong. The location of the root logical volumes is stored in the **boot data reserved area** (BDRA). The boot data reserved area contains the locations and sizes of LVM disks in the root volume group and other vital information to configure the root, primary swap, and dump logical volumes, and mount the root file system.

The BDRA contains the following records about the system's root logical volumes:

- Timestamp, indicating when the BDRA was last written
- `checksum` for validating data.
- Root volume group ID.
- Number of LVM disks in the root volume group.
- A list of hardware addresses of the LVM disks in the root volume group, and indices into that list for finding root, swap, and dump.
- Information needed to select the correct logical volumes for root, primary swap, and dumps.

Information about the LVM disk data structures in the BDRA is maintained by using the *lvlnboot*(1M) and *lvrmboot*(1M) commands. Here is sample output, followed by explanation:

```
# ./lvlnboot -v -d /dev/vg00/lvol2
Boot Definitions for Volume Group /dev/vg00:
Physical Volumes belonging in Root Volume Group:
                /dev/dsk/c3d0s2 -- Boot Disk
                /dev/dsk/c4d0s2 -- Boot Disk
                /dev/dsk/c5d0s2
                /dev/dsk/c12d0s2 -- Boot Disk
Root: lvol1     on: /dev/dsk/c3d0s2
                    /dev/dsk/c4d0s2
Swap: lvol2     on: /dev/dsk/c3d0s2
                    /dev/dsk/c4d0s2
Dump: lvol2     on: /dev/dsk/c3d0s2, 0
```

9

- The physical volumes (LVM disks) designated "Boot Disk" are bootable, having been initialized with `mkboot` and `pvcreate -B`.

- Multiple lines for `lvol1` and `lvol2` reveal that the root and swap logical volumes are being mirrored.
- Notice that `lvol2` is being used for both swap and dump, but that mirroring applies to only swap.

LIF Information

Much like non-LVM boot disks, the LVM boot disk contains a LIF volume, in which are stored `ISL`, `HPUX`, `AUTO`, `IOMAP`, `RDB`, and `LABEL`.

The LIF `LABEL` file is created and maintained by `lvlnboot` and `lvrmboot`. The `LABEL` file has pointers to the starting point and size of boot-relevant logical volumes, including the file system containing the kernel. With information in `LABEL`, HPUXboot can locate and gain access to the logical volume for root. The support and install tapes also use the information to access the root, primary swap, and dump logical volumes without actually using LVM.

Caution If the LIF or any reserved area becomes corrupted, the support tape can be used to recover the operating system. If that fails, or is impractical, the system must be re-installed.

The LIF volume cannot be copied directly onto LVM disk using *dd*(1M). It must be set up by the install process or by *mkboot*(1M), on LVM disks for which the `pvcreate -B` option was used, due to a gap between the LIF header and LIF directory. When using *mkboot*(1M), you should specify section 2, not section 6.

Physical Volume Reserved Area (PVRA)

Everything LVM needs to know about an LVM disk is contained in the **physical volume reserved area** (PVRA). The physical volume reserved area stores information about the LVM disk configuration and operation, starting addresses, and sizes. It also contains a bad block directory for the LVM disk.

The PVRA contains the following record of the LVM disk in relation to its volume group:

- LVM identification field, signifying that the LVM record is present and valid.
- Unique physical volume ID, checked by the LVM when it generates new physical volumes.
- Physical volume number within the volume group.
- Last physical sector number, used in determining the amount of space available on the disk.
- Size of each physical extent on the LVM disk, expressed exponentially in `DEV_BSIZE` blocks. All physical extents are identical in size for all disks in a volume group.
- Amount of space (expressed in number of `DEV_BSIZE` blocks) allocated for each physical extent on the LVM disk.

The PVRA also contains the starting addresses (physical sector number and length) of the following areas:

- Boot Data Reserved Area (for boot volume groups)
- Volume Group Reserved Area
- Volume Group Descriptor Area
- Volume Group Status Area
- Mirror Consistency Records
- User Data Area
- Bad Sector Relocation Pool

The bad block directory of the PVRA contains record of all software sectors that are faulty, have been relocated, or are awaiting relocation. Hardware sectors are not included.

Volume Group Reserved Area (VGRA)

The **volume group reserved area** (VGRA) describes the volume group to which the disk belongs. This information is organized in three subareas:

- Volume Group Descriptor Area (VGDA)
- Volume Group Status Area (VGSA)
- Mirror Consistency Record

Volume Group Descriptor Area (VGDA)

The **volume group descriptor area** (VGDA) contains the information the LVM device driver needs to configure the volume group for LVM, including:

- Volume group header with timestamp to indicate when the VGDA was last updated, volume group ID number, and three configurable operating system parameters (see *System Administration Tasks* manual for further information):
  - `maxlvs`—maximum number of logical volumes allowed per volume group.
  - `maxpxs`—maximum number of physical extents allowed per LVM disk.
  - `maxpvs`—number of LVM disks that can be installed in the volume group.
- List of logical volume entries, one for each logical volume within the volume group, recorded as follows:
  - Maximum number of logical extents permissible per logical volume.
  - Current status and capabilities of the logical volume.
  - Mirroring schedule policy, if set.
  - Maximum number of mirror copies allocated, if set.
- List of LVM disks, including:
  - Header with global information (ID, number of physical extents, status) about the disk.
  - Map of physical extents to logical volumes.
- Volume group trailer, timestamped when the VGDA was last updated. (Both header and trailer timestamps are compared to verify the consistency of the VGDA.)

The VGDA is the area checked to ensure a quorum of total LVM disks in the volume group. A volume group cannot be activated unless half the disks in a volume group are present and available, that is, containing the latest state information. The `vgchange -q n` option can be used to override the system's quorum check.

Caution Overriding quorum can result in a volume group whose configuration is inaccurate (for example, missing recently creating logical volumes). This configuration change might not be reversable.

Since each physical extent is recorded in a VGDA entry, the extent size has a direct bearing on the VGDA. In most cases, the default extent size will work perfectly well. However, if you run into problems, you might consider the following circumstances:

- Since the VGDA is a fixed size, a high-capacity LVM disk might exceed the total number of physical extents allowed. As a result, you might need to use a larger-than-default extent size on high-capacity LVM disks.
- Conversely, if all LVM disks in a volume group are small, the default number of extents might make the VGDA too large, wasting disk and memory space. A smaller-than-default extent size or number of physical extents might be preferable.
- A high-capacity LVM disk might be unusable in a volume group whose extent size is small or set with a small number of physical extents per disk.

Volume Group Status Area (VGSA)

The **volume group status area** (VGSA) tracks the validity and availability of LVM disks and the current state of physical extents (stale or fresh). The LVM reads this information when it initializes the volume group, updates it as a result of configuration changes to the volume group, or as a result of I/O errors.

The VGSA can be considered an extension of the VGDA, because it duplicates some of VGDA information, such as quorum, `maxpvs`, and `maxpxs`.

Mirror Consistency Record (MCR)

When the OSF Mirror Write Cache is used, the **mirror consistency record** (MCR) minimizes the amount of I/O required to bring all mirrors into a consistent state following a system crash or power failure.

The MCR contains records of:

- Timestamp when the mirror consistency record was last written.
- Minor numbers of the logical volumes involved.

- Number and size of the logical track group to which the logical volume is associated.

User Data Area

The user data area is the region of the LVM disk used to store all user data, including file systems, virtual memory system (swap), or user applications.

When you create the volume group, you apportion the user data area into fixed-size physical extents. Physical extents map to logical extents that hold user data—file systems, virtual memory subsystem, or a user application.

By default, physical extent size is 4 MB. On high-capacity LVM disks, the default limit on the total number of physical extents per LVM disk might necessitate a larger extent size, or a larger number of physical extents per LVM disk.

Bad Block Relocation Policy

Bad blocks can cause serious data transfer problems. The LVM device driver checks each physical request for bad blocks in the address range of the request. If it encounters a bad block on a mirrored LVM disk, LVM can recover from the error by reassigning the data to other blocks on the disk. This feature, called **bad block relocation**, is enabled by default.

With this feature, LVM supports two types of bad block relocation:

- Soft relocation, which assigns a new block to replace the defective one.
- Hard relocation, in which LVM instructs the disk device driver to spare the bad sector(s).

The following commands deal with LVM bad block relocation policy:

lvcreate(1M)	Sets bad block relocation policy for logical volumes being created in a given volume group.
lvchange(1M)	Allows you to change the policy to prevent or allow bad block relocation for logical volumes in a given volume group.
pvcreate(1M)	Lets you specify the minimum number of bad blocks that LVM should reserve to perform software bad block relocation. (By default, one block for every 8K of data blocks is reserved.)

It is good practice to run diagnostics testing for bad blocks on any device before initializing an LVM disk.

LVM Overhead

The data structures that enable you to use LVM consume a modest amount of overhead from your disk space. This overhead is set at a fixed boundary (2912 KB) for bootable LVM disks and may vary in size in non-bootable LVM disks (typically 400 KB).

Overhead required by non-bootable LVM disks depends on the parameters listed below, which can be modified using SAM or the **`vgcreate`** command. If you set a small extent size or create many physical volumes, your LVM data structures (overhead) will be larger.

`-e`	Sets the maximum number of physical extents allocatable for LVM disks in a volume group (by default, 1016).
`-l`	Sets the maximum number of logical volumes allowed in a volume group (by default, 255).
`-p`	Sets the maximum number of LVM disks (physical volumes) allowed in a volume group (by default, 32).
`-s`	Sets the size, in megabytes, for each physical extent in a volume group (by default, 4).

An operating-system parameter, **`maxvgs`**, can be configured to set the maximum number of volume groups LVM will recognize. Its default (10) is in the **`/etc/master`** file and can be overridden (permissible range is 0 to 256) by adding a statement to the `S800` file.

Understanding LVM Configuration Maintenance

While LVM provides desirable flexibility on your HP-UX system, it also adds a level of complexity for system administration. Be sure that you safeguard your LVM configuration.

Guidelines for Configuring the Root Volume Group

The following options are recommended when setting up the root volume group using *lvcreate*(1M) command. The same conditions can be applied when using SAM:

`-C y`	Makes extents contiguous.
`-r n`	Turns off bad block relocation.

For any swap logical volume, the following options should be used:

`-C y`	Makes extents contiguous.
`-M n`	Turns off the Mirror Write Cache (used for mirroring).
`-c n`	Turns off all synchronizaton.

A dump logical volume requires only the following special options, as long as it is not also a swap logical volume, in which case it requires the same options as swap.

`-C y`	Makes extents contiguous.

The /etc/lvmtab File

At the heart of the LVM configuration is the `/etc/lvmtab` file, which is read by all LVM commands. `/etc/lvmtab` is not readable or editable on-screen.

The `/etc/lvmtab` file is run-time generated; that is, it is generated the first time you create an LVM entity using SAM or commands such as `vgcreate`, and updated every time you change the LVM configuration. Every configuration update or query reads the `/etc/lvmtab` file.

LVM disk file names are recorded in `/etc/lvmtab`. If `/etc/lvmtab` file is destroyed, all configuration operations involving LVM data structures become impossible.

You can recover the `/etc/lvmtab` file using `vgscan`. To safeguard against LVM configuration problems, use the *vgcfgbackup*(1M) and *vgcfgrestore*(1M) utilities to back up and restore the LVM configuration.

Note `vgcfgbackup` and `vgcfgrestore` back up and restore *only* the LVM configuration, not the user data the LVM data structures contain. You still have to back up and restore user data using HP-UX utilities, such as `fbackup` and `frestore`.

The `vgcfgbackup` command backs up volume-group configuration information into binary files, one file per volume group.

The `vgcfgrestore` command restores LVM disk configuration in case of disk loss or data corruption.

`vgcfgbackup` backs up the following:

- LVM record of each LVM disk.
- BDRA record for each bootable LVM disk.
- One copy each of the current VGDA and VGSA data structures.
- LIF `LABEL` files.

`vgcfgbackup` does not back up the following:

- LIF header and files
- bad block directory.

Another command useful in maintaining the `/etc/lvmtab` file is *vgscan*(1M). If `/etc/lvmtab` is deleted or corrupted, running `vgscan` recreates the `/etc/lvmtab` file. `vgscan` searches every LVM disk on the system for logical volumes, then groups them into volume groups by searching the `/dev` directory and matching major and minor numbers.

Dealing with Removable Media and Volume Groups

Before a volume group created on another computer system can be accessed, the volume group must be associated with its new system by being written into `/etc/lvmtab`. Two commands, *vgexport*(1M) and *vgimport*(1M) deal with these circumstances.

The `vgexport` command removes the definition of a volume group from the system (`/etc/lvmtab` and device files) without removing it from the disks. This allows the disks to be brought to another system and used there (after you execute `vgimport`).

The `vgimport` command scans a specified list of LVM disks, rebuilds volume group information, updates `/etc/lvmtab`, and sets up LVM special files for the new volume group.

LVM Maintenance Mode

Maintenance mode boot provides a means, during system installation, or when critical LVM data structures have been lost, to boot from an LVM disk without the LVM data structures.

After the support tape has made the LVM disk minimally bootable, the system can be booted in maintenance mode using the `-lm` option to the `hpux` command at the `ISL>` prompt. This causes the system to boot to single-user mode without primary swap, dump, or LVM to access the root file system.

For further information, refer to the manual, *Installing and Updating HP-UX* or *hpuxboot*(1M) in the *HP-UX Reference Manual.*

Caution The system *must not* be brought to multi-user mode (that is, `init 2`) when in LVM maintenance mode. Corruption of the root file system might result.

Mirroring

Note Mirroring requires the optional product, HP MirrorDisk/UX, part number B2491A.

Mirroring is not available on HP-IB disks.

Mirroring is the capability of storing identical copies of data on separate LVM disks or on separate locations of the same disk. This redundancy has several advantages:

- The system can survive LVM disk crashes if you mirror the root file system and swap.
- Valuable data is available on more than one LVM disk. This benefit is known as high availability. If an I/O channel fails, LVM can recover the data from the duplicate source.
- Mirror-write recovery mechanisms enable the system to synchronize data.
- Mirroring speeds read-intensive applications by enabling the hardware to read data from the most convenient LVM disk, thus optimizing I/O.
- Backups can be done on one copy of the data while another copy continues to run.
- Mirroring of data to areas of the same LVM disk allows you to overcome local media errors.

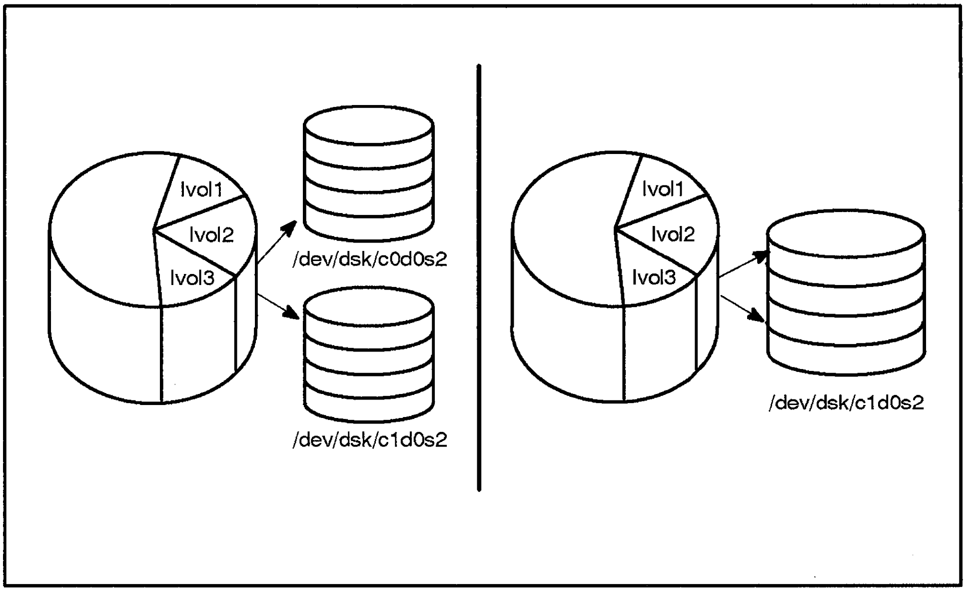
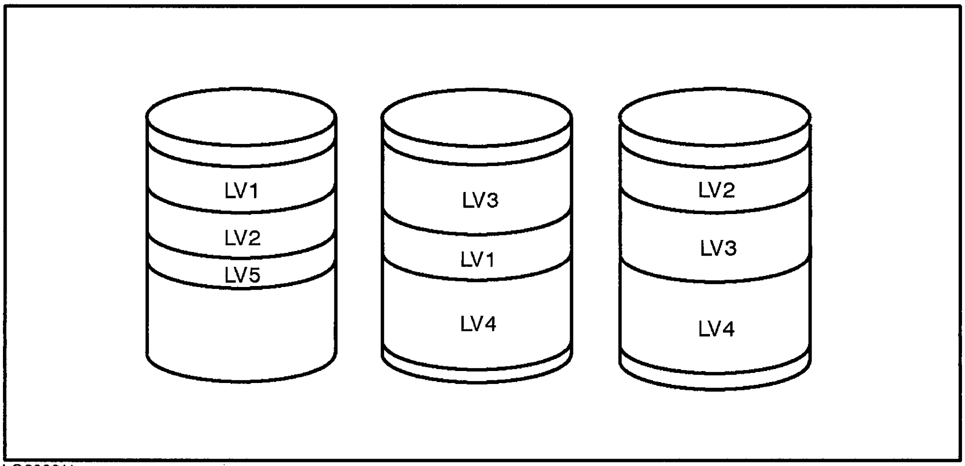


Figure 9-14. Strict Mirroring of Logical Volume lvol2

Figure 9-14 shows logical volume `/dev/vg00/lvol2` mapped to physical extents on LVM disks `/dev/dsk/c0d0s2` and `/dev/dsk/c1d0s2` (left side) or to different areas of physical extents on the same LVM disk, `/dev/dsk/c1d0s2` (right side).

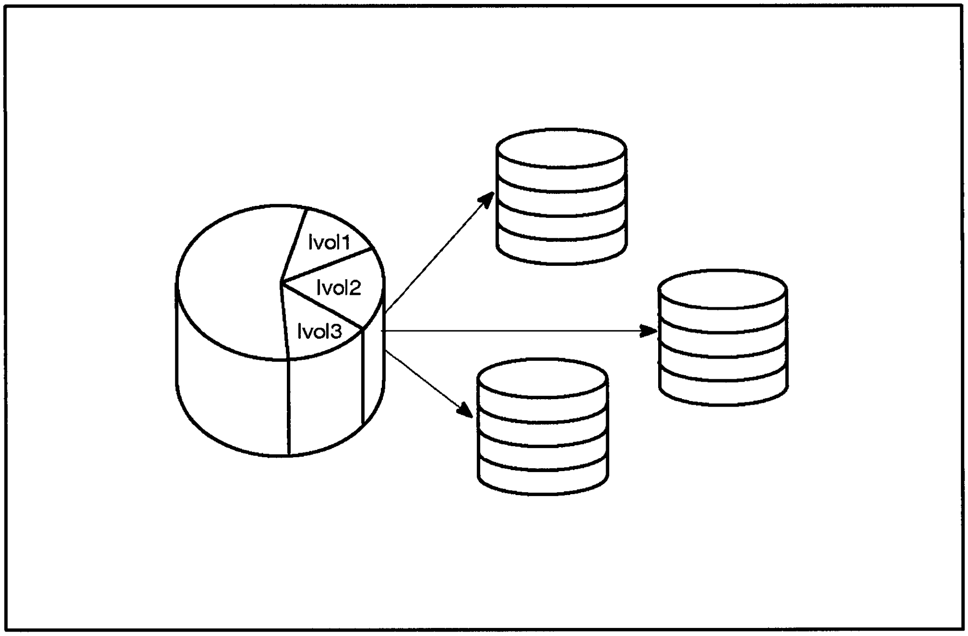
In Figure 9-15, you can see how mirrored logical volumes (`LV`) might distribute over three LVM disks. (In this example, all logical volumes but `LV5` are mirrored.)



LG200211_xx

Figure 9-15. Three LVM Disks with Mirrored Logical Volumes

Both examples in Figure 9-14 demonstrate **single mirroring**. That is, in both cases, `lvol2` maps from *one* logical volume to *two* sets of physical extents on LVM disks. Another kind of mirroring is possible. **Double mirroring** maps *one* logical volume to *three* sets of physical extents on LVM disks. (Sets of physical extents can map strictly to separate LVM disks or non-strictly to different areas of the same disk or disks.) Double mirroring is shown in Figure 9-16.



LG200211_033

Figure 9-16. Three LVM Disks Mirror a Single Logical Volume

Each copy of mirrored data maps to the same logical volume; the number of logical extents remains constant, but the number of used physical extents (and therefore, occupied disk space) changes, depending on the number of mirrored copies.

Mirrored logical volumes must belong to the same volume group; you cannot mirror across volume groups.

I/O Channel Separation

I/O channel separation is an approach to LVM configuration requiring that mirrored copies of data reside on LVM disks accessed via separate device adapters (interface cards) and cables. I/O channel separation achieves higher availability and better performance by reducing the number of single points of possible hardware failure. If you mirror data on two separate disks, but through one card, you are vulnerable to failure if the card fails.

You can separate I/O channels on a system with multiple disk adapters but a single bus (such as HP Model 857S) by mirroring disks across different interface cards. This is shown in Figure 9-17, in which the LVM disks at addresses `44.0.0` and `48.0.0` are accessed through different I/O channels (slots `11` and `12`) on the HP-PB bus.

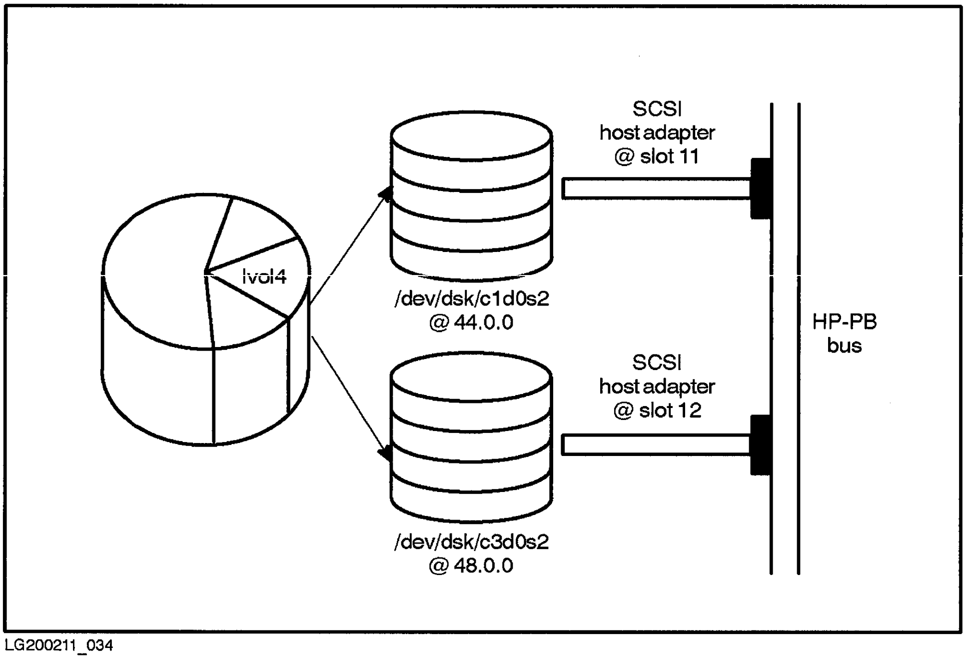


Figure 9-17. I/O Channel Separation via Separate Interface Cards

9

On systems such as the Model 870S, you can mirror disks across separate buses, as shown in Figure 9-18. The logical extents (`LE1`, `LE2`, and `LE3`) in memory map to the physical extents (`PE1`, `PE2`, and `PE3`) in LVM disks labeled `PV1` and `PV5`.

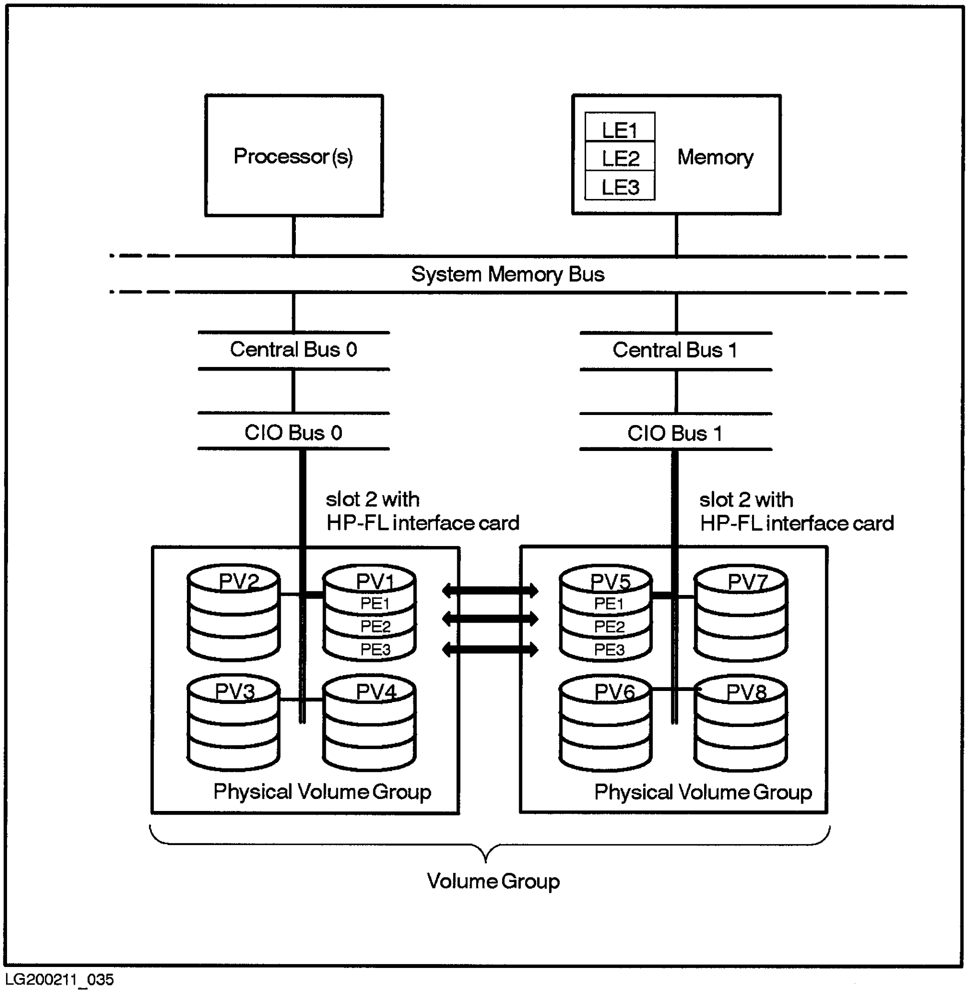


Figure 9-18.
I/O Channel Separation via Separate Buses, and using Physical Volume Groups

You can further ensure channel separation by establishing a policy caled **PVG-strict allocation**, which requires logical extents to be mirrored in separate **physical volume groups**. Physical volume groups are subgroups of LVM disks (physical volumes) within a volume group. An ASCII file, `/etc/lvmpvg` contains all the mapping information for the physical volume group, but the mapping is not recorded on disk. Physical volume groups have no fixed naming convention; you might name them `PVG0`, `PVG1`, and so on, or something more intuitive. The `/etc/lvmpvg` file is created and updated using the `vgcreate`, `vgextend`, and `vgreduce` commands.

I/O channel separation is particularly useful for databases, because it heightens availability (LVM has more flexibility in reading data on the most accessible logical extent), resulting in better performance. If you define your physical volume groups to span I/O devices, you ensure against data loss, even if one card fails.

For maximum performance, group the LVM disks by type (for example, all HP-FL or all SCSI). When using physical volume groups, you might also want to use a PVG-strict allocation policy for logical volumes.

As with other mirroring features, physical volume groups are not supported on HP-IB.

How Mirrored Logical Volumes are Written

Three sets of policies govern how mirrored logical extents are written to physical extents:

- Allocation policy.
- Scheduling of disk writes.
- Synchronization policy for crash recovery.

Allocation, scheduling, and synchronization can be set using SAM, *lvcreate*(1M), or *lvchange*(1M).

Allocation Policies for Mirroring

Mirrored data can be distributed on LVM disks by strict or non-strict, contiguous or non-contiguous allocation. By default, allocation of mirrored logical volumes is set to strict, non-contiguous.

- Strict allocation requires logical extents to be mirrored to physical extents on different LVM disks.
- Non-strict mirroring allows logical extents to be mirrored to sets of physical extents, not necessarily on different LVM disks.
- Contiguous allocation policy has three characteristics:
  - Physical extents are allocated in ascending order.
  - No gap exists between physical extents within a mirror copy.
  - All physical extents of a mirror copy reside in a single LVM disk.
- Non-contiguous allocation allows logical extents to be mapped to non-consecutive physical extents.

When the root volume group is mirrored, it must be set up with contiguous extents. That is, the physical extents used by logical volumes for root, primary swap, or dump devices must consecutive for all mirrored copies.

Scheduling Policies for Disk Writes

The LVM scheduler converts logical requests into one or more physical requests, then schedules them through the physical layer. Scheduling occurs for both mirrored and non-mirrored data.

To ensure maximum control, two I/O schedule policies are available—parallel and sequential.

- **Parallel**. The parallel scheduling policy is used by default with mirroring for maximum I/O performance. Parallel scheduling causes mirror operations to write simultaneously to all copies. LVM performs reads in an optimized fashion, by reading from the LVM disk with the fewest outstanding I/O operations.
- **Sequential**. The sequential policy causes mirror write operations to proceed sequentially; that is, one write completes before the next write begins. Likewise, LVM mirrors are read in a predefined order. On a practical level, sequential policy would be used only for extreme caution in maintaining consistency of mirrors.

Synchronization Policies for Recovering Mirrored Data

Maintaining consistency of mirrored data can be accomplished by configuring your system using any of three different approaches:

- Mirroring with the OSF Mirror Write Cache enabled.
- Mirroring with the OSF Mirror Write Cache disabled and LVM Mirror Consistency Recovery enabled.
- Mirroring without using an LVM mirror consistency mechanism.

The OSF Mirror Write Cache provides a fast resynchronization of data following a system crash or failure, but at a potential performance cost for routine system use.

The OSF Mirror Write Cache keeps track of all available space on the LVM disk to which you are writing, periodically recording the activity in an on-disk data structure called the Mirror Write Consistency Record. An extra disk write is required for every mirrored write not already marked on the LVM disk. This slows down on-line processing and degrades performance when you access the disk at random; when writing to an area of the disk that is already marked, performance is not impaired. Upon system reboot after crash, the operating system uses the relatively small record of the Mirror Write Cache to resynchronize inconsistent data blocks quickly.

The frequency of extra disk writes is small for sequentially accessed logical volumes (such as database logs), but increases when access is more random. Therefore, logical volumes containing database data or file systems with few or infrequently written large files (greater than 256 KB) should not use the OSF Mirror Write Cache when run-time performance is more important than crash-recovery time.

When mirroring with the OSF Mirror Write Cache disabled and LVM Mirror Consistency Recovery enabled, LVM does not impact run-time I/O performance. However, for any logical volumes using Mirror Consistency Recovery, the entire data space is resynchronized when the volume group is activated. Synchronization is performed in the background without interfering with reboot or access; however, while during this time, I/O performance is degraded. This approach is useful if optimal run-time I/O performance is required.

When mirroring *without* using any LVM mirror consistency mechanism, the operating system's run-time behavior is identical to that of the previous approach. This approach is useful when running an application program (such as a database) that has its own means of maintaining or recovering consistent data, such as transaction logfiles. Note, however, that the database logfiles themselves can be configured as a mirrored logical volume to use effectively the OSF Mirror Write Cache.

Manual Synchronization.
An HP-UX command, *lvsync*(1M), can be used to manually refresh extents that have been marked "stale." This approach recovers "stale" extents by reading from a "current" mirrored physical extent and writing to the "stale" physical extent.

Backing Up Mirrored Data

Backups can be performed on one copy of an off-line logical volume, while another copy is operational. This is done using the `lvsplit` command to take a mirrored logical volume off-line, backing it up, then using `lvmerge` to return the backup copy of the logical volume on-line. Double mirroring (mirroring onto three LVM disks) enables you to continue mirroring while performing the backup. (This is especially useful for high availability environments.)

Useful Mirroring Commands

Although you can perform all mirroring tasks using SAM, the commands listed below deal with synchronization and scheduling in a mirrored environment. Their manual pages, in *HP-UX Reference*, provide further information about LVM capabilties. Procedures for the tasks to which they are related are 9 found in Chapter 8, "Managing Logical Volumes" of the Series 800 *System Administration Tasks* manual.

lvcreate(1M)	Creates a logical volume in a volume group, at which time you can set the scheduling policy, set the number of mirror copies, enable or disable the OSF Mirror Write Cache, and enable or disable the LVM Mirror Consistency Recovery mechanism.
lvchange(1M)	Changes the characteristics of a logical volume. This includes scheduling policy, enabling or disabling the Mirror Write Cache, enabling or disabling the Mirror Consistency Recovery mechanism.
lvextend(1M)	Increases the number of physical extents allocated to a logical volume.
lvdisplay(1M)	Shows information about logical volumes, including the mode set for mirror consistency recovery and scheduling policy.
lvsync(1M)	Synchronizes physical extents that are stale in one or more mirrored logical volumes.
vgchange(1M)	Sets volume group availability to the logical volume manager.
lvsplit(1M)	Divide a mirrored logical volume into two logical volumes.
lvmerge(1M)	Combine two logical volumes into one logical volume.

Internal Representation of LVM

Figure 9-19 illustrates the relationship between the LVM subsystem and the HP-UX system architecture in general:

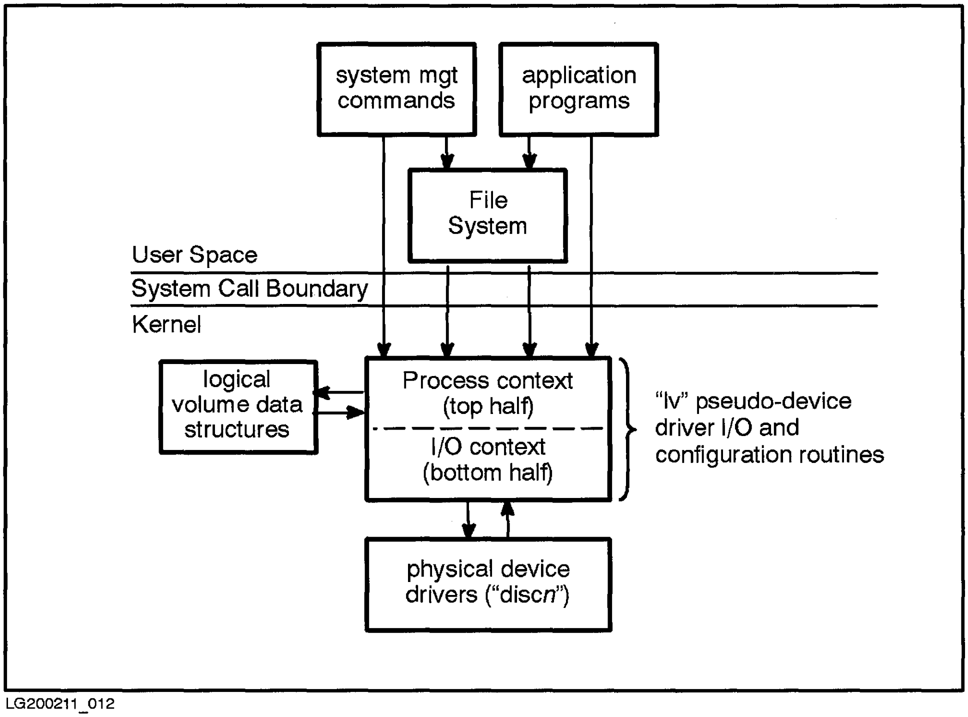


Figure 9-19. Architecture of the LVM Subsystem

The LVM pseudo-device driver (`lv`) handles all I/O operations for the LVM components on behalf of applications, file systems, and other subsystems.

System management commands perform LVM configuration operations by opening the volume group's control device and issuing LVM `ioctl` calls. When applications issue `open`, `read`, `write`, `ioctl`, and `close` calls, the top half of `lv` does everything required to send a request to the underlying disk drivers. This includes ensuring that requests do not overlap, choosing how to access the mirrors, translating logical addresses to physical addresses, and calling the driver to start the I/O.

The bottom half of the LVM handles everything associated with request completion from the underlying disk driver. This includes finding locating a mirror if one is not already available, restarting requests previously blocked due to overlap, restarting requests waiting for the Mirror Write Cache to be written, and other functions.

The bottom half executes in interrupt context; thus, it is always interrupting user processes. However, since it is not running in the context of a process, the LVM cannot access user process information. It also should complete as quickly as possible and cannot go to sleep (for example, to wait to allocate memory).

The `lv` pseudo-device driver maintains the on-disk data structures that describe the volume groups. Each volume group has a separate entry in the kernel device switch table, with entry points for the logical volume device driver and a pointer to the volume group data structure. Each volume group's control device file, called `group` (logical volume number 0), provides the means for manipulating the volume group's data structures. Each logical volume in the volume group is accessible through a device node with its own unique (non-zero) logical volume number.

To translate from a logical to physical operation, an `lv_xlate` function converts the logical address consisting of the volume group, logical volume, logical block, and mirror number, into a physical address consisting of physical volume and physical block. A mirror number and logical block index entry maps into the logical volume's extent map, which can then be used to find the LVM disk.

10

System Architectures

This chapter surveys the various HP-UX system platforms, with figures of card cages and bus structures. Visualizing the layouts of the specific architectures will help you understand HP-UX addressing and configuration.

Architecture of the Series 300 Bus

All HP 9000 Series 300 computers use either the DIO (Direct I/O) or DIO-II *internal* bus.

Buses are circuitry—the pattern of conductors (traces or wires) on a circuit board—through which data travels between hardware modules (CPU, memory, channel adapters, device adapters) within the system. Cards are plugged into these buses.

The number of data conductors equals the number of data bits of the bus. A 16-bit DIO bus has sixteen conductors; a 32-bit DIO-II bus has 32 conductors. The 16-bit DIO bus transfers data at I/O bandwidths of 3 MB/second; the 32-bit DIO-II bus transfers data at 6 MB/second.

DIO and DIO-II cards come in various sizes:

- DIO system cards are designed in pairs. The upper card is designated as the accessory card and the lower is the I/O card. (One exception: the Model 340 has a single DIO I/O card, and no accessory slot.)
- DIO system cards for processors or video interfaces are twice as wide as DIO I/O and accessory cards.
- DIO-II cards are similar in size to the DIO system card, but include a 32-bit backplane connection. Only DIO-II cards are supported in DIO-II slots, unless a card is identified as supported in both types of slots. DIO adapters are available for backward compatibility.
- Multiplexer (MUX) cards can be plugged into an internal DIO or DIO-II slot to connect terminals, modems, printers, plotters, and other RS-232C devices.
- Daughter boards are, strictly speaking, not interfaces; instead, they screw onto cards to add RAM on a processor or RAM board, additional interface on the system interface board, or an accelerator on a video board.

Both Model 350 and 370 workstations have an additional internal RAM bus, fitted as a top plane opposite the DIO-II connector. The RAM bus provides higher speed, and therefore better performance, by bypassing the DIO-II bus, and thus avoiding bus collisions with anything running on the backplane.

External buses (high speed and standard-speed HP-IB, VME, SCSI, and HP-HIL) connect peripheral devices to HP 9000 Series 300 computers.

Except for Models 318 and 319 workstations, and other lower-priced non-expandable units, you can enlarge your Series 300 memory or I/O capability by adding expander boards, either piggy-backed to the System Processor Unit (SPU) or HP-HIL card, or caged in a VME Expander bus connection. The VME Expander is supported on all DIO-II hosts.

Model 375—a DIO-II Implementation

The elements depicted in Figure 10-1 of a Model 375 workstation show the basic organization of its boards and is typical of Series 300 models.

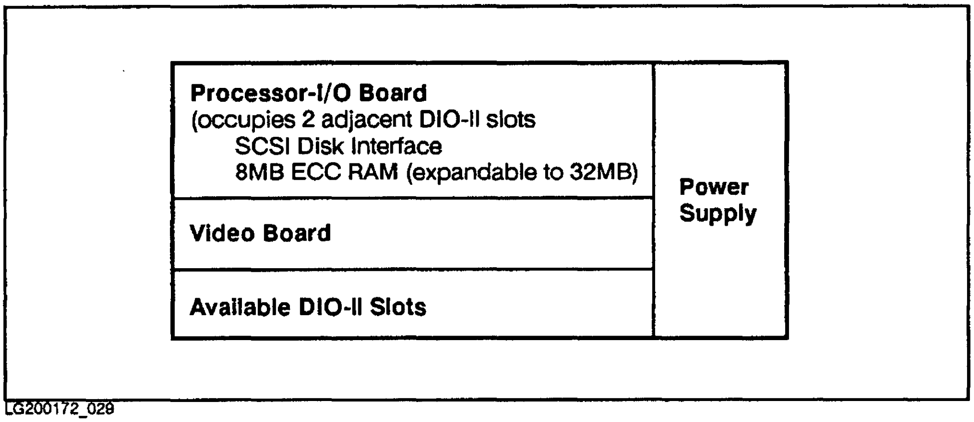


Figure 10-1. Backplane of a Model 375 Workstation.

Figure 10-2 depicts the functional relationship between the DIO-II bus on the Model 375 and its attached boards.

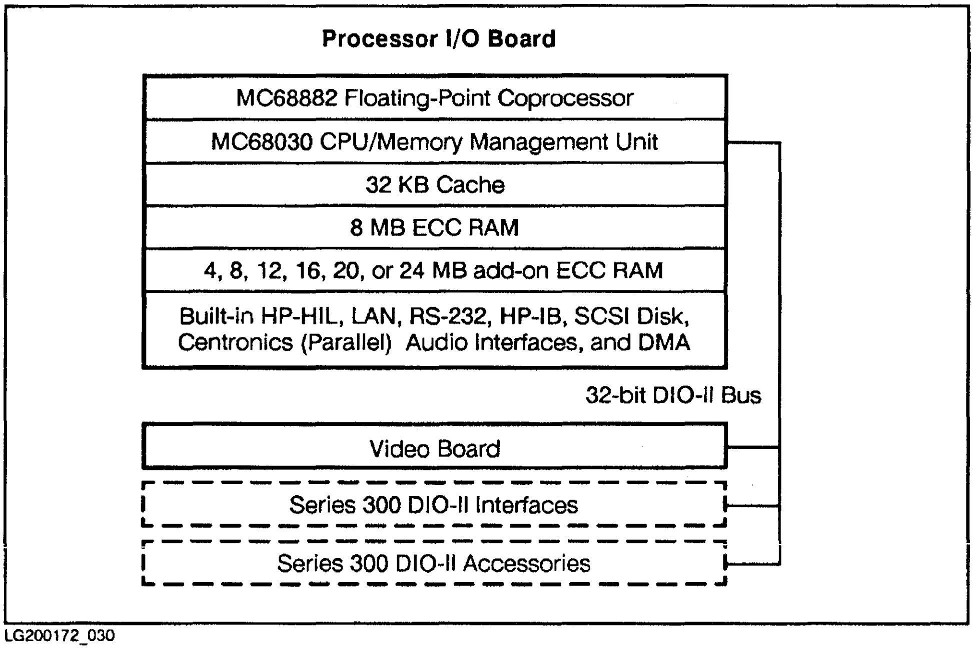


Figure 10-2. Model 375 Workstation Functions

To learn more about architectural differences and configuration possibilities, consult the *HP 9000 Workstations Configuration Guide* for each specific model.

Processor-I/O Board

The Processor-I/O board occupies one double-width slot on the DIO-II bus of the Model 375, and contains the following essential elements:

- Motorola MC68030 CPU/Memory Management Unit.
- MC68882 floating-point coprocessor.
- 32 KB cache (high-speed memory).
- Error checking and correcting RAM (minimum 8 MB). Small SIMM RAM cards mounted perpendicularly onto the CPU can increase the Model 375 to a maximum of 128 MB of RAM.

The following elements of the processor-I/O board are specifically relevant to configuration:

- SCSI (Small Computer System Interface) interface, used for connecting SCSI peripherals, including magnetic and optical disks, and digital data storage tape drives. SCSI is a very fast, industry-standard interface and an optional product for most Series 300 systems.
- Centronics interface (parallel) is used by peripheral devices, such as printers. Centronics is an eight-bit wide port capable of sending data through multiple (eight) lines instead of one. This simultaneous transmission results in faster data transfer than via the serial interface.
- High-speed HP-IB (IEEE-488) interface can be used to connect the System Processor Unit (SPU) to disk drives. In some systems, a SCSI interface replaces the high-speed HP-IB interface.
- RS-232C (serial) interface, used to connect a terminal, printer, plotter, or other serial device to Series 300 systems.
- LAN interface, which enables you to hook up to a local area network.
- Audio output, which connects to an external speaker.
- HP-HIL (Human Interface Link) interface, used to hook up the keyboard and other HP-HIL devices, such as a mouse, digitizer, and button box.

Other interfaces are available, including the standard-speed HP-IB interface, used to connect numerous devices, including plotters, instruments, and tape drives.

Graphics Interface

The video board provides the interface for the monitor. A number identifies the video board, and should be used to set terminal type. Setting terminal type incorrectly (or too vaguely) results in application products (such as `vi`) working less efficiently than intended.

The correct terminal type is complicated by the issue of how you will be using your terminal. If you plan to work in raw mode, HP recommends that you identify the actual product number of your display (look for it near the HP logo on the front or back panel) or execute `/etc/dmesg`). The terminal type (`TERM`) corresponds to an entry in `/usr/lib/terminfo`, where the system looks for information on terminal resolution, number of lines per screen, and other monitor specifics.

To reference the proper terminal type, enter the following for Bourne or Korn shells:

```
$  TERM=98550
$  export TERM
```

or for C shell:

```
%1: setenv TERM 98550
```

This applies the settings for the current terminal session, but to make them permanent, you must add the same line to the `.profile` file for Bourne and Korn shells, or to `.login` for C shell.

Note If running VUE or X Windows, use generic `hp`, `hpterm`, or `hp300h` instead of the specific terminal type. This is because VUE, X Windows, and SAM have requirements that overrule specific terminal type.

Also, some terminal emulators (such as LAN configurations) require a different setting than indicated by the model number. The documentation of the terminal emulator takes precedence over these general guidelines.

A Graphics Accelerator, available on some color workstations (including 340CHX, 360CHX, 370CHX), replaces the video interface and approximately triples color graphics writing speed.

Architecture of the Series 400 Bus

Like the HP 9000 Series 300 Workstations, the HP Apollo 9000 Series 400 Workstations run HP-UX. The Series 400 workstations and servers include models 425e, 425t, and 425s, whose configurations include a variety of buses (including DIO-II and E/ISA) and networking capabilities, depending on model. The following block diagram depicts the basic configuration:

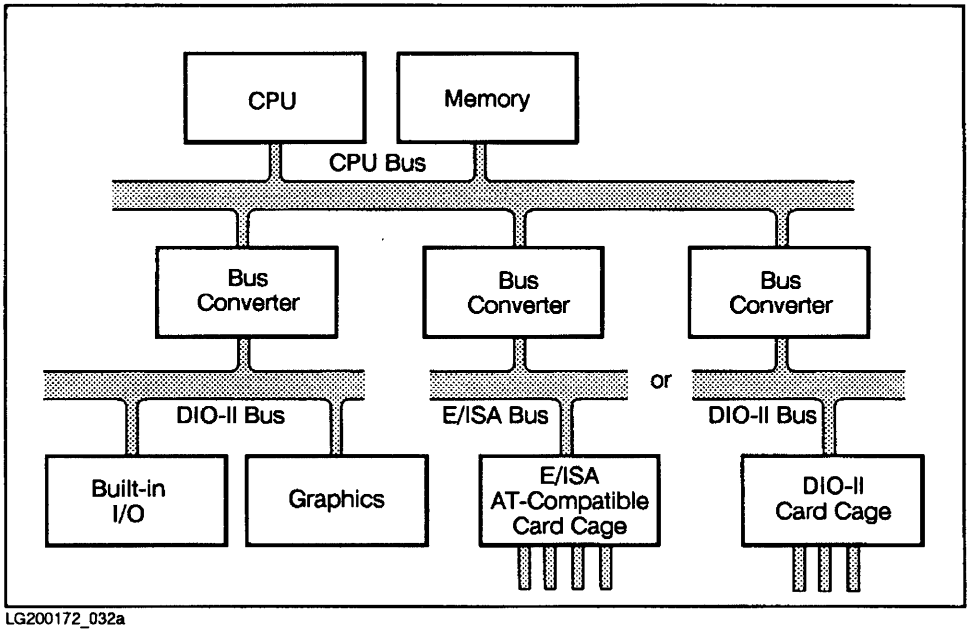


Figure 10-3. HP Apollo 9000 Series 400 Workstations Block Diagram

For information on the HP Apollo 9000 Series 400 Workstations, consult the installation guides, owner's manuals, and *HP Apollo 9000 Series 400 and Series 700 Pricing and Configuration Guide.*

Addressing on a Series 300/400

HP-UX finds devices connected to the bus by their **select code** and **bus address**. The executable `/etc/dmesg` displays for the root user the system's configured addresses, if run after boot-up.

Each select code and bus address uniquely identifies to the system an element of the architecture, as shown in Figure 10-4:

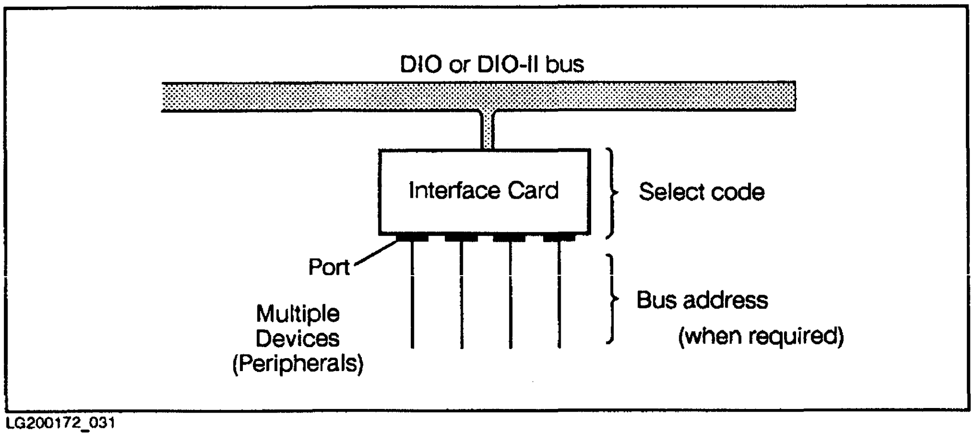


Figure 10-4. Series 300/400 Addressing

Series 300/400 Select Codes

All external devices connect to an interface card, either standard or optional. The interface card has a set of switches or jumpers, whose numeric value is referred to as its **select code**. The select code value is used to specify the port to which a device is connected, such as the LAN interface port shown in Table 10-1. The select code uniquely identifies the interface occupying a particular slot connected to the DIO or DIO-II bus.

Note Every card in the system must have its own select code. Cards cannot share select codes, even with other cards of the same type.

Table 10-1. Typical Series 300/400 Select Codes

Type of Interface	Preset Select Code	
	Decimal	Hexadecimal
High-speed HP-IB[1]	14	0x0e
SCSI[1]	14	0x0e
Centronics	12	0x0c
Standard-speed HP-IB	7	0x07
RS-232C (built-in)	9	0x09
MUX (4-port multiplexer)	13	0x0d
MUX (8-port multiplexer)	28, 29	0x1c, 0x1d
LAN	21	0x15
HP-HIL	0	0x00

1 If you have both high-speed HP-IB and SCSI interfaces, the two must have different select codes.

In the simplest case, the select code uniquely identifies the interface on the DIO or DIO-II bus. In the case of HP-IB, SCSI, RS-232C, MUX, or HP-HIL, when more than one peripheral can be connected to a single interface, the select code is used to identify the interface and a bus address distinguishes the specific device. In the case of multiple interfaces of the same type (such as two or more MUX cards), each card requires a unique select code.

Series 300/400 Bus Addresses

The interface card is placed in a slot in the backplane. The peripheral is connected to the card via a **port**, the place of access through which I/O comes into contact with the CPU. When an interface card is accommodating multiple peripherals, a **bus address** is necessary to distinguish between those multiple devices connected to a particular interface card.

Each device physically connected to an HP-IB, SCSI, or RS-232C MUX interface must be identified uniquely by a bus address. Setting the bus address may vary by device. For example, on the back of each HP-IB or SCSI device is a series of DIP switches or a rotary switch, which can be set to distinguish bus addresses between multiple devices.

Attaching to the MUX card is an automatic distribution panel with four or eight ports (numbered 0-3 or 0-7) into which a console or any serial device can be connected. This port number is analogous to a bus address and forms the second set of digits in the MUX hardware address.

The address of each daisy-chained HP-HIL device is automatically determined by its sequence, beginning from the HP-HIL port. Within the HP-HIL loop, many applications need the security ID module (part number 46084A) to run. If you include the security ID module in the loop, be sure to place it between the CPU and keyboard, and include it as one unit in the loop. Thus, for a configuration consisting of an ID module, keyboard, and mouse, the ID module is one, the keyboard address is two, and the mouse address is three.

Note Standard audio boxes do not have a bus address, even though they are in the HP-HIL loop.

Architecture of the Series 700 Bus

HP Apollo Series 700 workstations are designed around the PA-RISC 1.1 chip set for demanding graphics and computation-intensive applications.

Figure 10-5 shows rear views of Models 720 and 730 (left) and 750 and 760 (right). Other models conform to the same basic design.

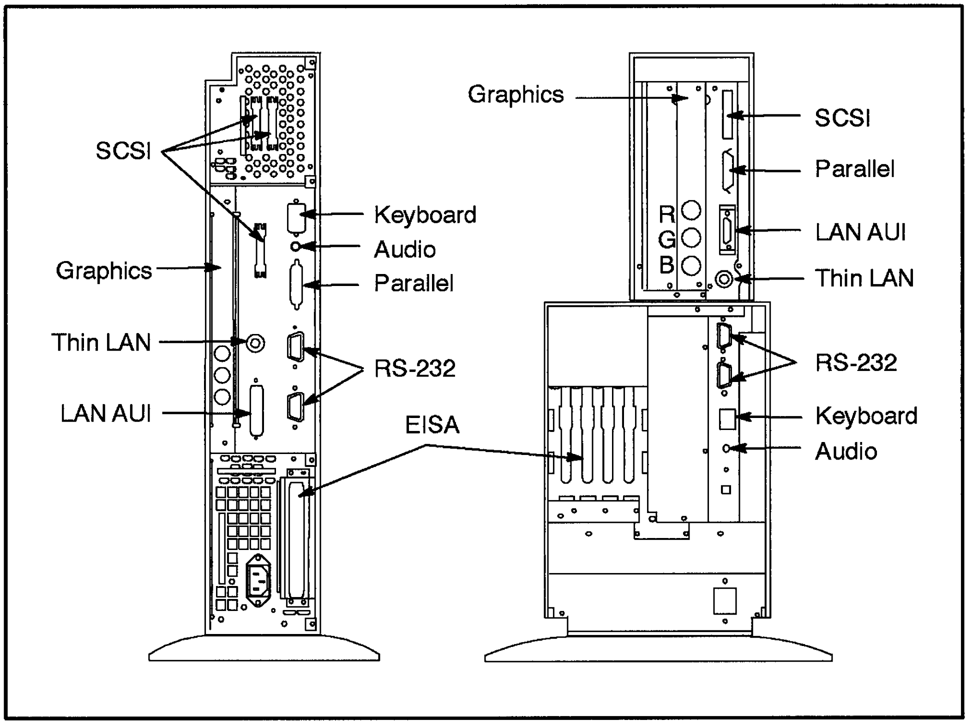


Figure 10-5. Models 720/730 and 750/760 Rear Panels, showing Ports

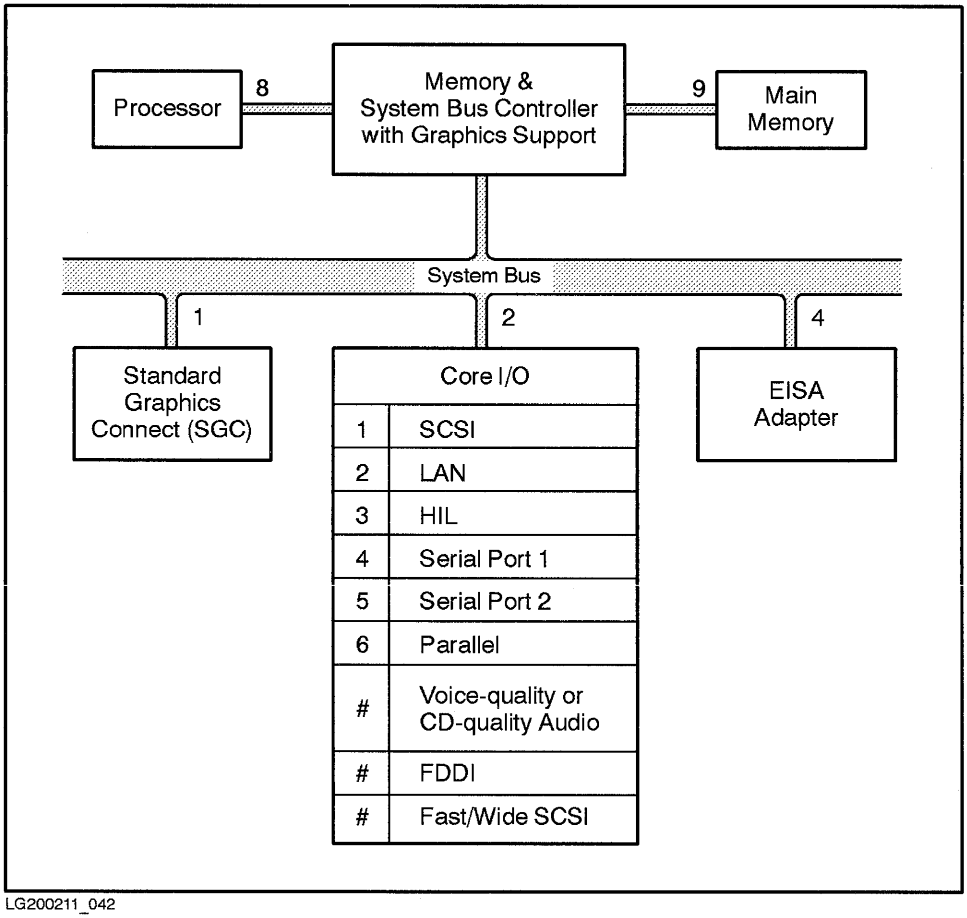


Figure 10-6. HP Apollo 9000 Series 700 Bus Architecture.

Figure 10-6 depicts the organization of the Series 700 bus architecture. Note that not all elements shown in the diagram are available on all systems, and addressing some elements of the core I/O varies by model. **#** symbolizes optional interfaces whose function number varies by model. For example,

- EISA is optional on Model 720.
- The number of graphics connection varies by model.
- Voice- or CD-quality audio is available only on Models 710 and 760.

- Systems may be equipped with either LAN or FDDI, but not both.

Series 700 Modules

Each module depicted in Figure 10-6 has a system bus module number (shown next to the bus); each interface of the core I/O also has a function number.

These bus module numbers and function numbers are used in device files and minor numbers for the Series 700. If you run `/etc/ioscan -f`, you will see these numbers displayed as part of the hardware path for each element of the module. Chapter 11, "System Configuration" discusses minor number construction in detail.

processor
: The processor module is implemented as a card with chips for CPU, data and instruction caches, clocks, and floating point coprocessor.

memory
: Depending on model, memory is implemented as SIMM cards in 8MB, 16MB, 32MB, and 64MB increments on the processor card, along with I/O-Memory control.

graphics
: The graphics adapter provides connectivity to color and grayscale monitors, 24-plane and 3D graphics subsystem capabilities. Voice- or CD-quality audio interface, plugged into the SGC (System Graphics Connect) bus, is available on some models.

core I/O
: Built-in I/O support is provided as a multi-functional card that includes HP-HIL, audio, two asynchronous RS-232C ports, IEEE 802.3 Ethernet LAN interface configured with ThinLAN, a Centronics parallel port, and an external SCSI-2 port. Each function is identified by a separate function number. Possible mass storage devices include 210MB, 420MB, 660MB, and 1.3GB disks, CD-ROM, floppy disk drive, and DAT (DDS) drive.

EISA
: An Extended Industry Standard Architecture (EISA) host adapter provides connectivity to I/O and networking cards, including EISA SCSI-2 (differential) host adapter, EISA HP-IB host adapter, EISA LAN/9000, FDDI/9000, X.25/9000, and IEEE 802.5 Token Ring 9000. The number of EISA slots varies by Series 700 model. EISA devices are not configured using SAM, but with the `eisa_config` utility.

Addressing on a Series 700

During boot-up, the boot ROM identifies devices currently attached to the system and their hardware paths. The boot ROM interface searches the architecture for a device having a bootable file system. If it encounters a cluster server, for example, it will give the LAN hardware address; if it finds a bootable disk along the SCSI chain, it will report its address. When running the boot ROM, if you press the (Escape) key, you get into the boot ROM interface. (See Chapter 2, "System Startup" for information.) You can see the output of the boot ROM by looking at `/etc/dmesg` discussed later in this chapter).

Interpreting Series 700 Hardware Paths using ioscan

The *ioscan*(1M) command is useful for visualizing the hardware configuration. The numbered modules shown in Figure 10-6 correlate to hardware paths displayed by `ioscan -f`. Here is a sample full listing on a Model 720:

```
Class    H/W Path  Driver   H/W Status    S/W Status
=====================================================
scsi     2.0.1     sim0     ok(0x7071)    ok
disk     2.0.1.0.0 scsi     ok(0x5800202) ok
disk     2.0.1.2.0 scsi     ok(0x101)     ok
disk     2.0.1.4.0 scsi     ok(0x800100)  ok
disk     2.0.1.5.0 scsi     ok(0x202)     ok
lan      2.0.2     lan01    ok(0x7072)    ok
hil      2.0.3     hil      ok(0x7073)    ok
serial   2.0.4     asio0    ok(0x7075)    ok
serial   2.0.5     asio0    ok(0x7075)    ok
parallel 2.0.6     parallel ok(0x7074)    ok
scsi     4.1.0     sim0     ok(0x7680)    ok
disk     4.1.0.6.0 scsi     ok(0x201)     ok
```

Multiple disks attached to the SCSI interface (hardware path `2.0.1`) are differentiated by the target number (for example, the `4` in `2.0.1.4.0`), and all the SCSI disks end in a terminating `0`. The disk addressed `4.1.0.6.0` is attached to an EISA host adapter at slot `1`.

Hardware-path numbering is directly related to minor numbers; compare the information in "Minor Numbers on the Series 700," in Chapter 11, "System Configuration".

Architecture of the Series 800 Bus

The Series 800 is built on a hierarchy of buses.

Buses are circuitry—the pattern of conductors (traces or wires) on a circuit board—through which data travels between hardware modules (CPU, memory, channel adapters, device adapters) within the system. Cards are plugged into buses.

The number of conductors used for data—the bandwidth—equals the number of bits of the bus. The fastest bus installed is used for data transfer between the CPU and memory. In the Models 8*x*7, private buses separate CPU and memory transactions from I/O, thus increasing system responsiveness, even for I/O-intensive computation. Among the I/O buses, HP Precision Bus (HP-PB) and CTB bus have bandwidths 32 bits wide; the CIO bus has a 16-bit bandwidth. Typically too, interface cards for graphics, communication, and factory-floor devices plug directly into the faster buses.

The Series 800 supports two overall bus architectures (HP-PB and CIO), with several variations. The CIO architecture has two implementations: a two-tier I/O bus structure (consisting of CIO and Mid-Bus) and a three-tier I/O bus structure (consisting of CIO, Mid-Bus, and SMB). The HP-PB architecture is used in numerous implementations, HP-PB alone and with private (processor) buses, and HP-PB with PMB. Table 10-2 shows the buses present in the various systems.

Table 10-2. Buses Supported on Series 800 Models

Acronym	Bus	Series 800 Models
HP-PB	Hewlett-Packard Precision Bus	808, 815, 8*x*2, 8*x*7, 890
CIO	Channel Input/Output	825, 834, 835, 840, 845, 850, 855, 860, 865, 870
SMB	System Main Bus	850, 855, 860, 865, 870
CTB	Mid-Bus or Central Bus	825, 834, 835, 840, 845, 850, 855, 860, 865, 870
PMB	Processor-Memory Bus	890

Depending on model, one or more of each bus is present. Thus,

- Models 808, 815, and 8*x*2 employ a single bus, HP-PB. The HP-PB provides direct connection between processor and memory modules and I/O cards.
- Models 8*x*7 employ the HP-PB bus for I/O and another set of busses for processor and memory transactions. Processor and memory communicate via subsidiary busses to a memory/I/O controller, which connects via the VSC bus to HP-PB.
- Models 825, 834, 835, 845, and 840 use a two-tier I/O bus structure of CIO bus and Mid-Bus. Models 825, 834, 835, and 845 can also be configured with an HP-PB converter for MAP applications.
- Models 850, 855, and 870 use a three-tier I/O bus structure of CIO bus, Mid-Bus, and SMB.
- Model 890 uses a two-tier bus structure: PMB for processor(s) and memory, and HP-PB for I/O.

Figures in coming pages illustrate these configurations.

Hardware modules—bus connectors, channel adapters, and device adapters—connect to these buses, and external peripherals attach to the device adapters. In the following section, each hardware module is discussed in turn, as well as its addressing.

Hardware Modules Connecting to the Buses

Hardware modules (bus converters, channel adapters, and device adapters) are cards or boards, inserted into slots in the system's card cage to connect to the bus architecture. The modules handle queueing, signal levels, and protocol when transferring data between the CPU or memory modules and specific devices (such as I/O and graphics) with distinctive signal needs. Once configured, transfer of data among bus structures is transparent in almost all respects. Each slot in the card cage is identified by a number or letter for hardware addressing.

The following hardware modules serve to connect buses or buses and devices:

Bus converter A board that serves as an internal interface between higher- and lower-speed buses, typically the SMB and Mid-Bus. Because more than one bus converter can be connected to the SMB, bus converters can increase the number of modules in a system, thus enabling more devices to be connected.

The Series 890 is equipped with dual bus converters: The top bus converter card resides in slots of the PMB (processor memory bus) and is attached by ribbon cable to one or two lower bus converters residing in HP-PB buses. (Each upper bus converter can actually provide bus conversion to two HP-PB buses.

Channel I/O adapter A module that serves as an internal interface between the Mid-Bus and the lower-speed CIO bus. The channel adapter also manages direct transfer of data between I/O modules and main memory with minimal CPU intervention. More than one channel adapter can be installed on most systems, thus expanding the number of possible modules.

Device adapter Interface cards such as HP-IB, HP-FL, SCSI, LAN, and MUX cards, installed in the I/O slots of the CIO or HP-PB bus. Device adapters make the crucial link that enables the system to communicate with external peripherals (such as terminals, printers, disk drives). Device adapters are also called I/O cards and host adapter cards. CIO device adapters are also called CIO cards.

Device adapters provide the data-transfer capabilities between the CIO or HP-PB bus and external devices (peripherals). Because of this, device adapters are the hardware modules most frequently configured by system administrators.

Common I/O interfaces include:

- HP-PB host adapters for HP-IB, HP-FL, and SCSI.
- HP-PB GPIO (general purpose parallel I/O), used for interfacing 8 or 16-bit data buses to the system. (Not supported on Models 8*x*7.)
- CIO multi-device interfaces for HP-IB, HP-FL, MUX, or SCSI.
- CIO system-to-system communication, such as LAN interfaces via IEEE 802.3 or Ethernet connection.
- I/O real-time interfaces, for faster response time than host systems routinely provide.
- Mid-Bus graphics interfaces, for interfacing display controllers to the system.
- Mid-Bus system-to-system communications interfaces, for communicating with X.25 packet-switching networks, other vendors' systems, or factory-floor devices.

For a complete listing of available interfaces, consult the *Configuration Guide* for your series.

SCSI, HP-FL, and HP-IB interfaces accommodate multiple devices per interface card; the devices are cabled together. With SCSI, you must cap the final device with a terminator plug. SCSI devices require an additional layer of addressing (described in the next section). If only one SCSI device is attached, the final character of the hardware address is 0.

When configuring HP-IB and other multiple-access interfaces, consider electrical characteristics, as well as what priority peripherals should have for accessing resources.

You need to understand major and minor numbers to configure HP-UX for a device. Refer to Chapter 11, "System Configuration" and to the *System Administration Tasks* manual for your system. Refer to the *Installing Peripherals* for complete installation and configuration instructions, including device-driver information.

Addressing on Series 800

Data is transferred through a sequence of each bus architecture's addressable elements, including bus location, bus converters, device adapters, and external devices.

At boot-up, the system reads the **hardware path** of the root disk as a sequence of addressable elements in the boot path from the CPU leading to the root disk device.

The same principles apply for any configured peripheral: The system recognizes the sequence of addressable elements in the I/O path from the CPU to the device.

Each addressable element is represented by a module number in the hardware path. The module number on both Mid-Bus and HP-PB bus is the actual slot number on the bus into which the card or board is plugged, multiplied by four. The multiplier of four is used to accommodate boards with multiple modules. On the CIO bus, the module number is the slot number.

In subsequent sections, addressing examples are given with each sample implementation. The addressable elements are symbolized as follows:

- `b` = bus converter address (PMB to HP-PB or SMB to Mid-Bus)
- `m` = module number (PMB, SMB, or Mid-Bus slot number times four; or HP-PB slot number times four plus unit number)
- `c` = slot address of a CIO interface card
- `d` = device switch setting
- `l` = logical unit number

Note that SCSI requires an additional "logical unit number" of `0` appended to the hardware address; if device is not SCSI, omit it. See *insf*(1m) in the *HP-UX Reference*.

The degree of addressing complexity depends on Series 800 model. The most complex example follows the pattern `b/m.c.d.l` for the Model 850 family and Model 890. HP-PB Models 8x2 and 8x7 are simpler to address, following the pattern `m.d` or `m.d.l`.

HP-PB Implementations—Models 8x2

The HP Precision Bus (HP-PB) enables data transfer through Models 808, 815, and $8x2$ via a single bus. CPU board (including processor and optional floating-point processor), memory modules, and all device adapters attach directly to the HP-PB.

Figure 10-7 depicts the card-cage layout for a Model 822S system, as seen from the rear:

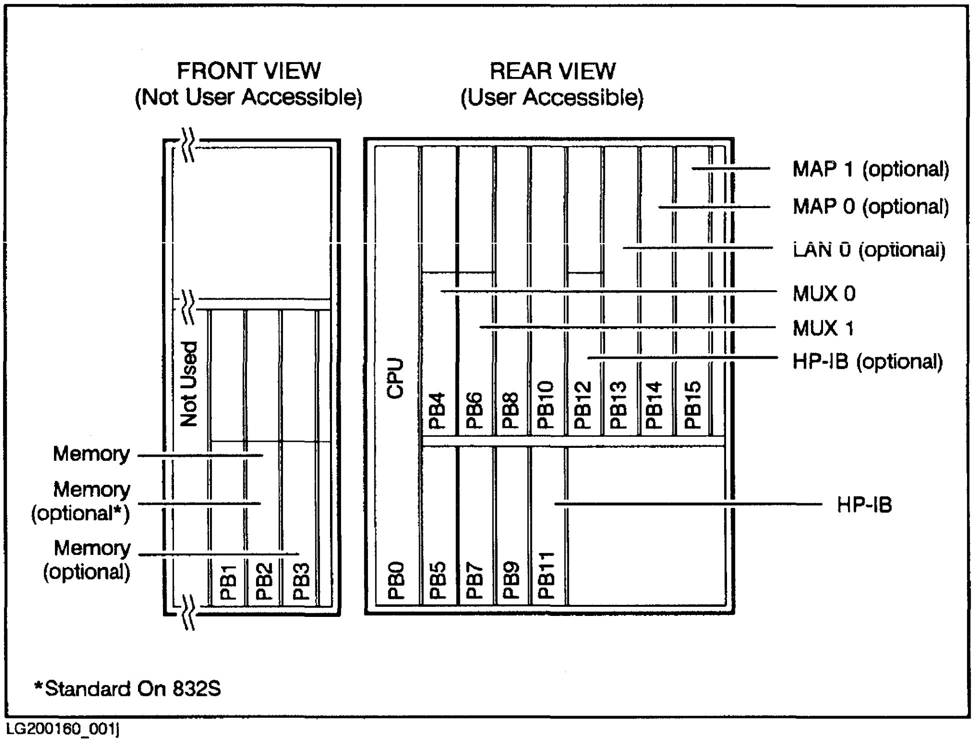
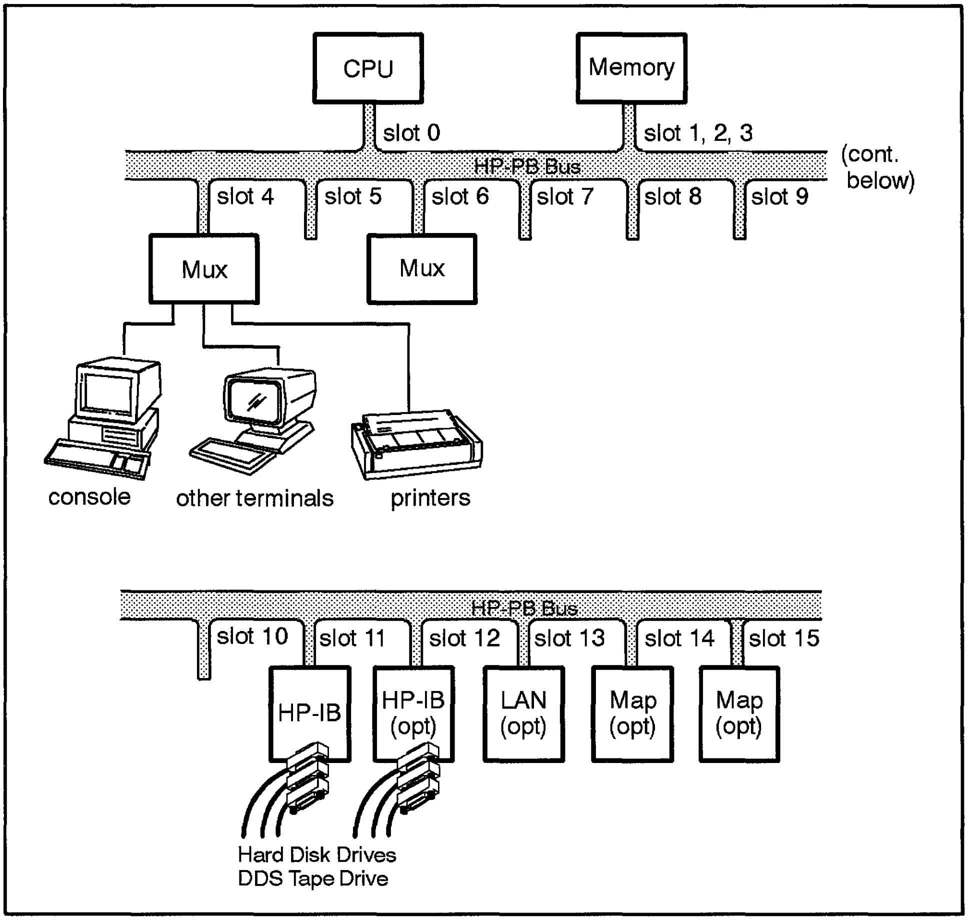


Figure 10-7. Sample Model 8x2S Card-Cage Layout

Figure 10-8 depicts the bus architecture and hardware modules configured in the Model 8x2S:



LG200172_033a

Figure 10-8. Model 8x2S bus architecture

Addressing on HP-PB Models

I/O interface cards plug directly into the Precision Bus, making hardware addressing relatively simple. The hardware address (`m.d`) consists of two elements:

- `m` = module number (HP-PB slot number times four)
- `d` = device switch setting

The following are found in standard configurations:

- Slot 11 (module 44) is used for an embedded hard disk drive and DDS tape drive. Thus, a DDS tape drive with switch set to 7, connected to the HP-IB interface at slot 11 has the address `44.7`.
- Slot 12 (module 48) is used for attaching external HP-IB devices.
- The console must be attached to the first port on the first MUX (in this case slot 4, module 16); each MUX can provide from five to sixteen ports and can have other serial devices (such as terminals and printers) attached.
- If installed, a LAN card occupies slot 13 (module 52) by default.
- When present, Manufacturing Automation Protocol (MAP) cards are usually installed in slots 14 and 15 (modules 56 and 60) for control of factory-floor devices.

For more information about configuring HP-PB models, refer to the *Installing Peripherals* and *System Administration Tasks* manuals.

A Model 8x7 Implementation

The Model $8x7$ bus architecture consists of a bus architecture similar to the Model $8x2$ family, but with several refinements, including faster processor and memory access. Figure 10-9 shows the backplane of a Model 857S/877S/897S.

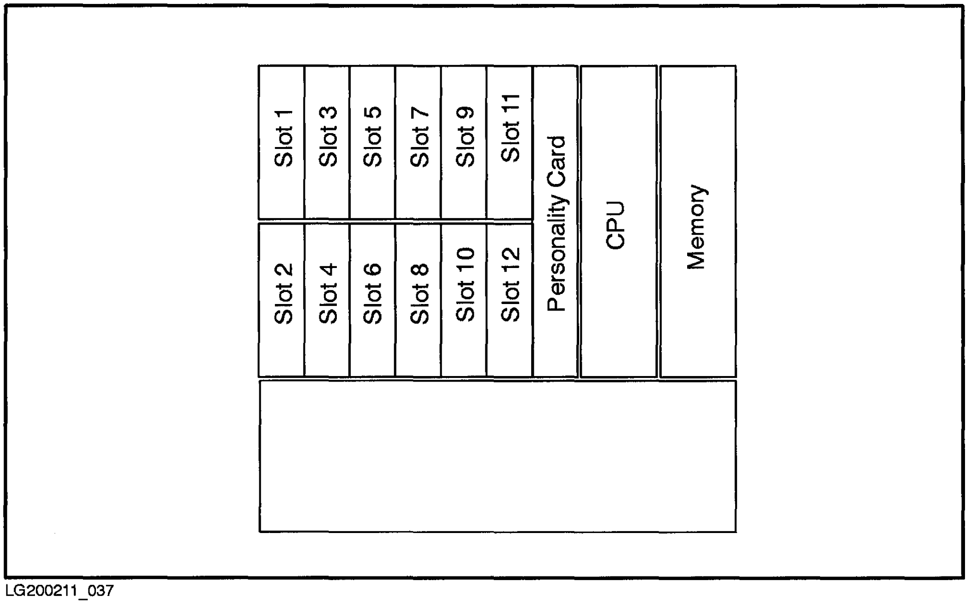


Figure 10-9. A typical backplane of a high-end Model 8x7S

Figure 10-10 shows the bus architecture of a typical Model 8x7. Note that processor and memory interactions are segregated from the HP-PB bus, minimizing I/O contention.

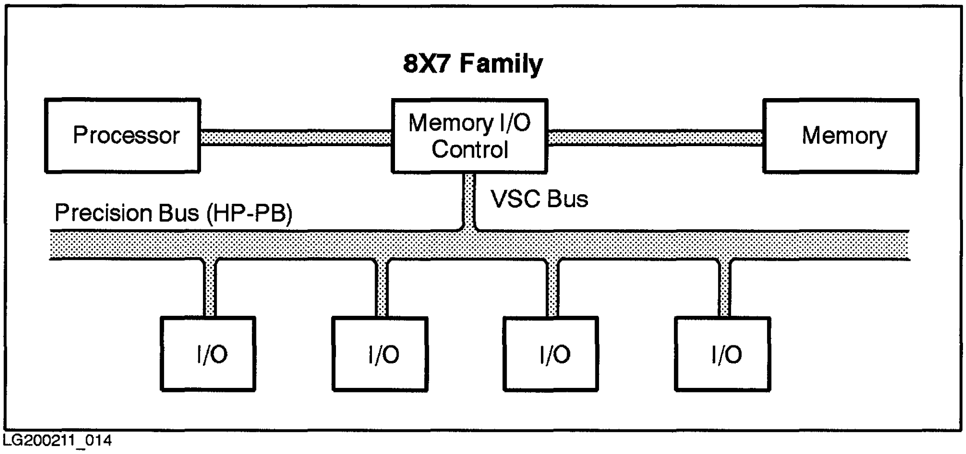


Figure 10-10. Model 8x7S Bus Architecture

In Figure 10-11, you can see the module numbers used to address each element of the bus architecture.

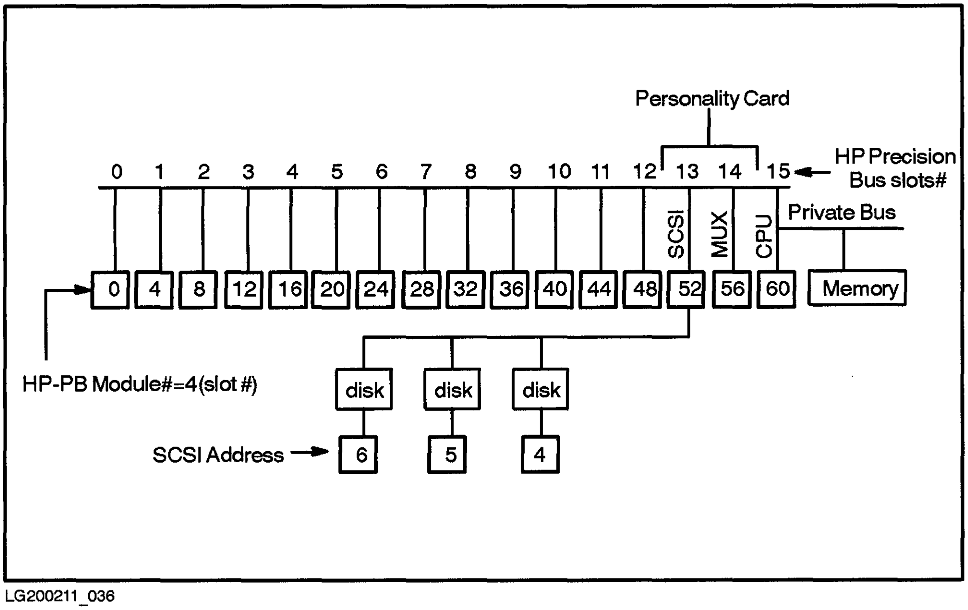


Figure 10-11. Addresssing the Modules of a Model 8x7S

Addressing Models 8x7

Addressing the Model 8x7 is virtually identical to that of the Model 8x2, in that I/O interface cards plug directly into the Precision Bus. The hardware address (`m.d`) consists of two elements:

- `m` = module number (HP-PB slot number times four)
- `d` = device switch setting

If a SCSI device (`s`) is attached, append an extra `0` (`m.d.s`).

Models 8x7 feature multi-functional "personality" cards whose addressing deserves some attention. For example, the standard personality card consists of 8 to 16 (optional) RS-232 ports, console/access port, parallel/Centronics port, and SCSI port. As shown in Figure 10-11, the personality card is plugged into HP-PB slots 13 and 14 (modules `52` and `56`). Module number `52` has two units—`0` for SCSI and `1` for Centronics. Module number `56` is used for MUX connections. The following addresses are derived:

`52.6.0`	SCSI root disk
`53.0`	Centronics printer (not supported on Model 807)
`56.0`	console

A Mid-Bus/CIO Implementation—Model 845SE

The two-tier bus architecture of the Model 845 (Figure 10-12 and Figure 10-13) is representative of a family of systems that include Models 825S, 835S/SE, and 845S/SE.

Addressing on Mid-Bus/CIO Models

Hardware addressing on Mid-Bus models typically requires a three-part hardware address (`m.c.d`) to take into account multiple bus structures of Mid-Bus and CIO bus.

- `m` = module number (Mid-Bus slot number times four) of the channel adapter
- `c` = slot address of the interface card in the CIO bus
- `d` = device switch setting

If a SCSI device (`s`) is attached, append an extra `0` (`m.c.d.s`).

Boards on the Model 845 plugged directly into the Mid-Bus are identified by module number, Mid-Bus slot times four. Thus, a memory board connected at slot 8 (default position) has a module number of `32`.

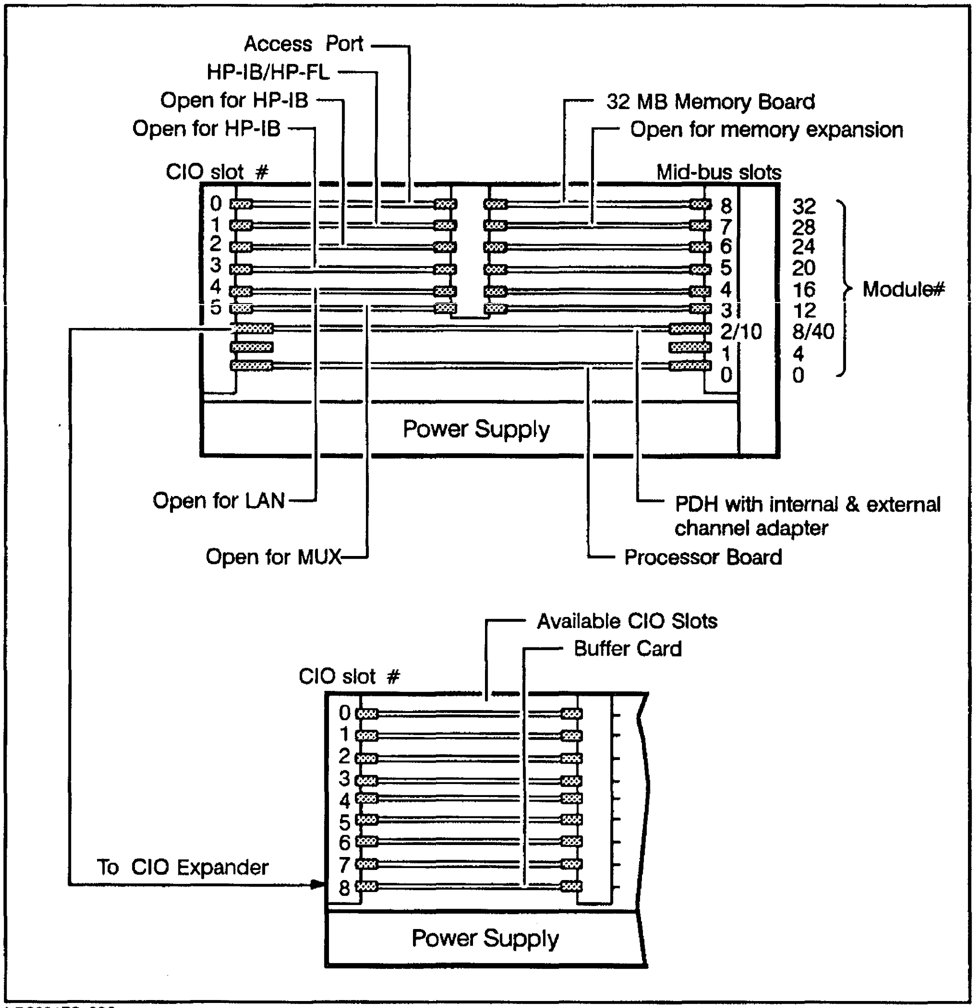
Addressing the processor-dependent hardware (PDH) is slightly different, for two reasons: the board occupies two mid-bus slots (2 and 10), and has two channel adapters (one for each CIO bus—one internal and one external). Each channel adapter is addressed separately and may have module number `8` or `40`, depending to which CIO bus the channel adapter refers.

Another addressing variation occurs on Model 835, where the channel adapter chip set is located directly on the system card, plugged into Mid-Bus slot 1. In this case, the channel adapter chip set has a module number `4`.

To address peripherals (attached to CIO interface cards) requires addressing the channel adapter module, the interface card at its CIO slot, and the device setting itself. In Figure 10-13, a disk connected at the first port (port 0) of the HP-IB at CIO slot 1 is addressed `8.1.0`.

Factory-Floor Communication

An alternative configuration enables Mid-Bus models to handle computer-aided manufacturing. The HP Precision Bus (HP-PB) is added to a Mid-Bus system through an HP-PB bus converter. This enables installation of Manufacturing Automation Protocol (MAP) cards (IEEE 802.4) for control of factory-floor devices. Figure 10-12 depicts the card-cage layout of a Model 845SE, with attached CIO expander.



LG200172_036

Figure 10-12. Sample Model 845 card-cage layout

Figure 10-13 depicts the bus architecture and hardware modules configured in the Model 845 system:

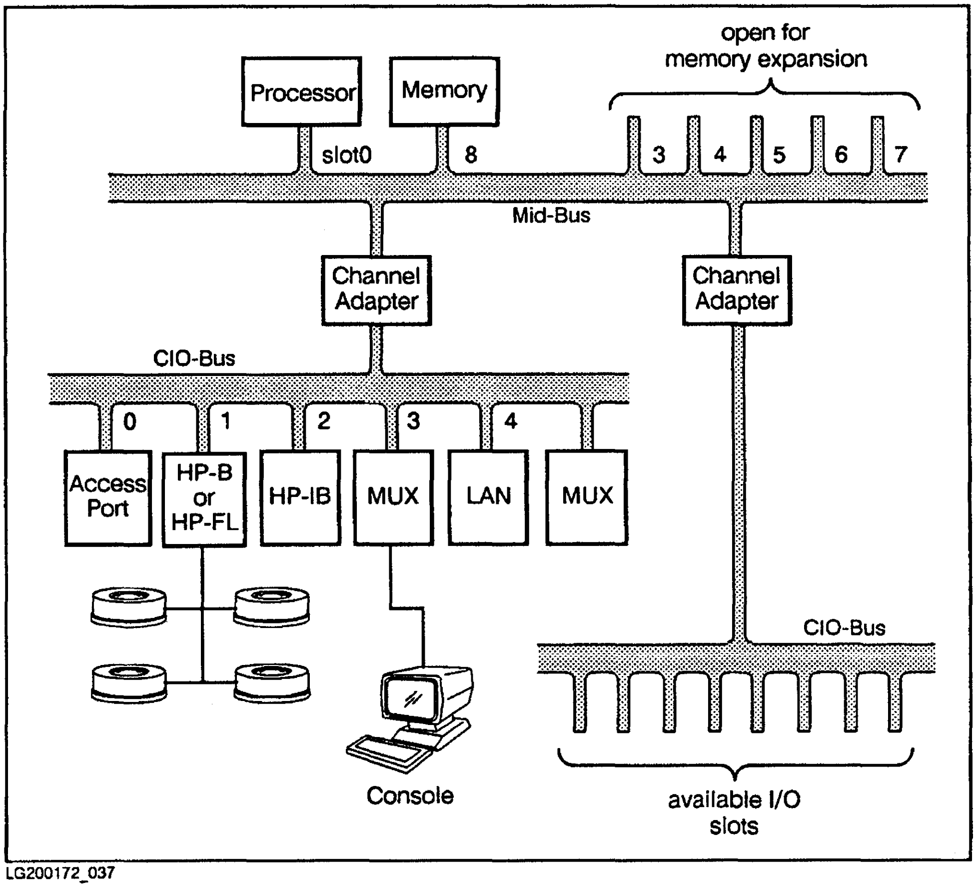


Figure 10-13. Model 845 Bus Architecture

An SMB Implementation—Model 850S

Models 850S, 855S, 860S, 865S, and 870S are large systems built on a hierarchy of three buses—SMB, Mid-Bus, and CIO. This arrangement enables high-capacity operation via numerous devices.

At 9.0, Model 870S can be equipped with up to four processors.

Figure 10-14 depicts the card-cage layout of a Model 850S.

Addressing on SMB Models

Hardware addressing on SMB models must descend through all the elements in the architectural hierarchy. Thus,

- Devices attached to the Mid-Bus are addressed by specifying the bus converter between SMB and Mid-Bus and module into Mid-Bus.
- Devices attached to the CIO bus are addressed by specifying the bus converter between SMB and Mid-Bus, channel adapter between Mid-Bus and CIO, CIO card slot, and device setting.

The hardware address of a device configured into an SMB system follows the pattern `b/m.c.d`, which corresponds to:

- `b` = bus converter address (SMB to Mid-Bus—either `2` for bus converter 0 or `6` for bus converter 1)
- `m` = module number (Mid-Bus slot number times four) of the bus adapter
- `c` = interface card (the actual CIO slot number)
- `d` = device switch (set on the device itself)

Figure 10-15 depicts the bus architecture and hardware modules configured in the Model 850S system. For example, the tape drive connected to an HP-IB port has the address `2/4.2.3`.

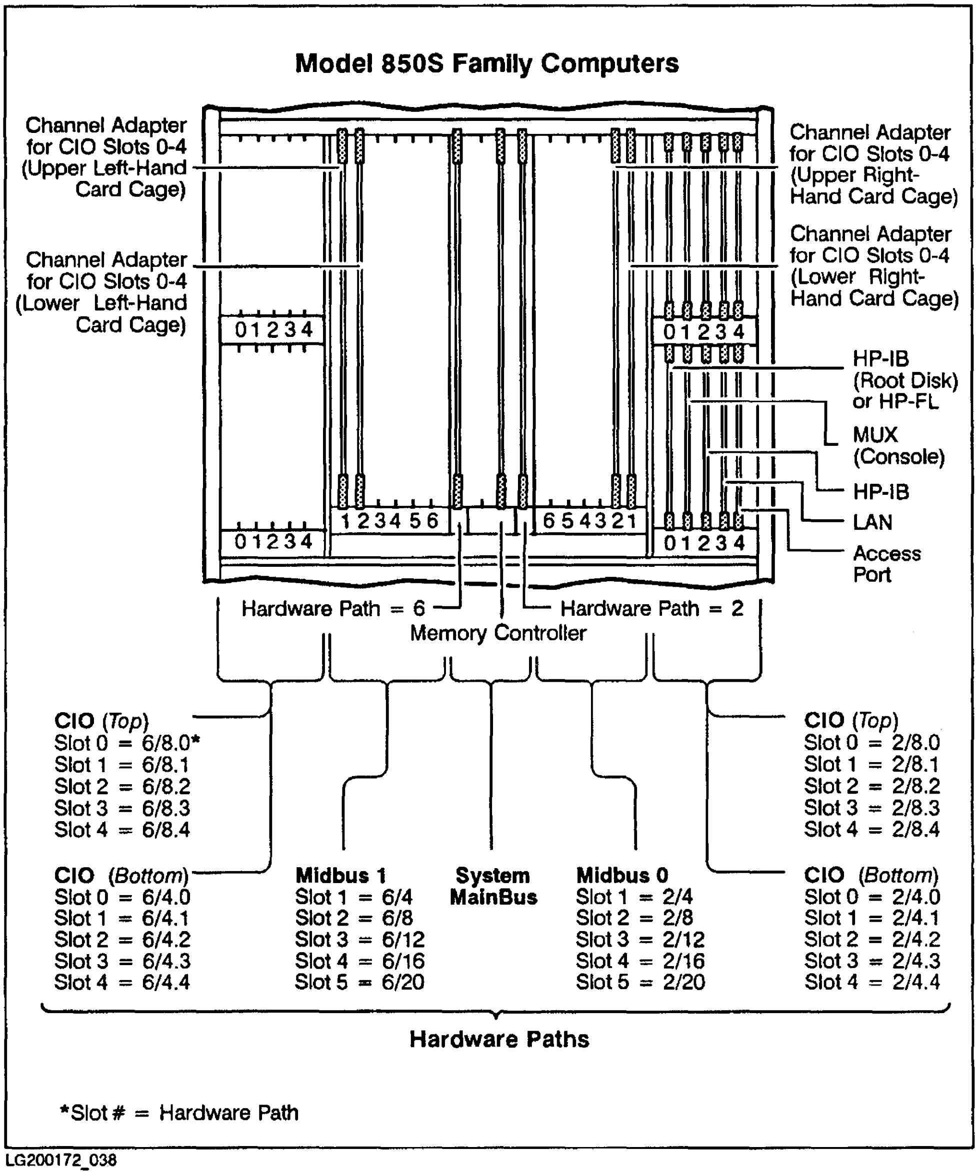


Figure 10-14. Sample Model 850S Card-Cage Layout

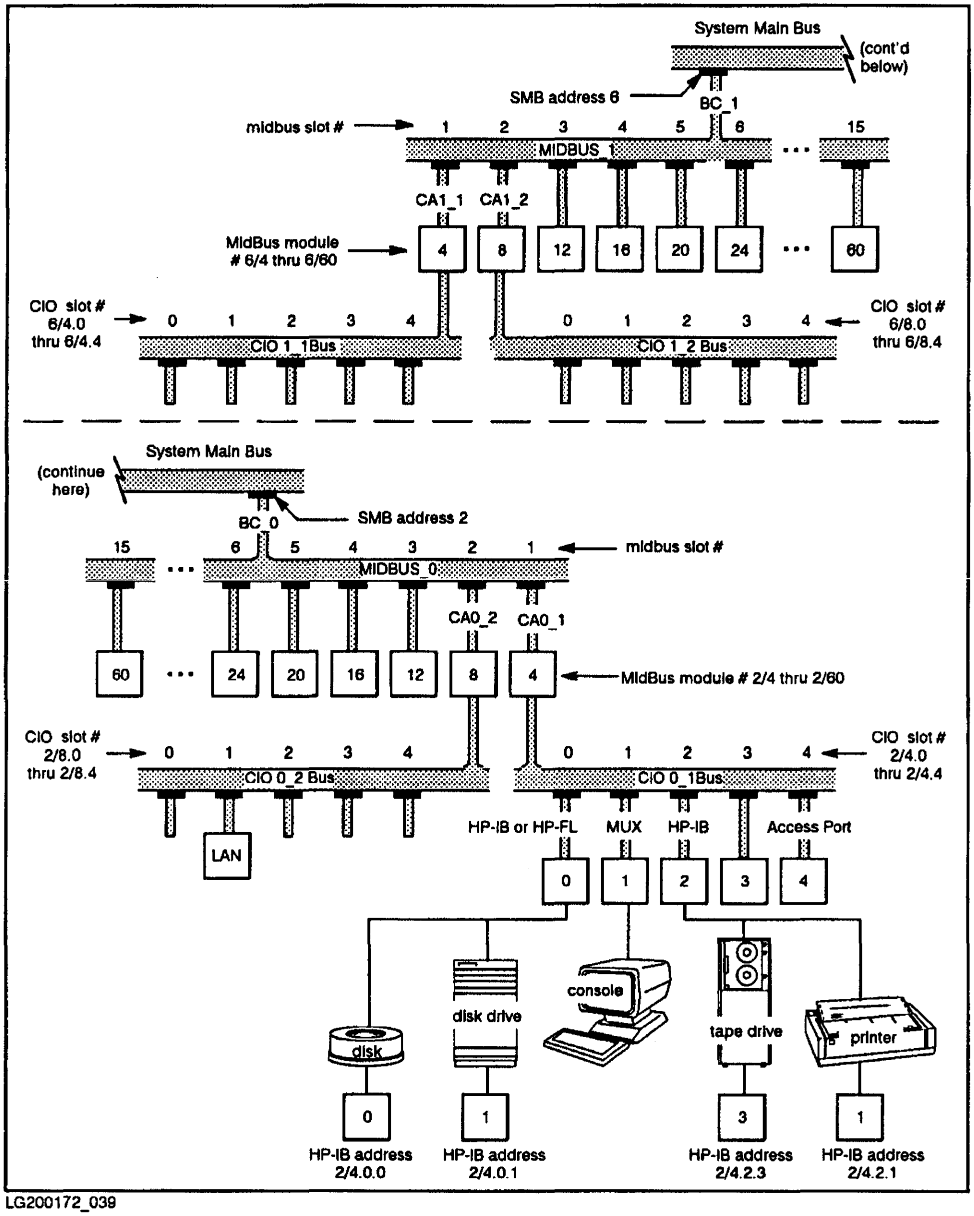


Figure 10-15. Model 850 bus architecture

Processor-Memory Bus (PMB) Implementation—Model 890

The Series 9000 Model 890 Corporate Business Server is the high-end HP-UX computer. Architecturally, its organization is hierarchical, like SMB systems, but using the HP-PB (instead of CIO) bus.

- Like the Models 870 and 880, the cardcage of the Model 890 can be outfitted with up to four processors, to perform multi-processing. Configuration of the multiple processors should be handled by an HP customer engineer.
- Like the Models 8*x*7, memory communicates with the processor(s) via a bus structure separate from I/O. This ensures that extremely fast processing time, even on I/O-intensive systems.

Physical Layout of the Model 890

As shown in its simplest form, in Figure 10-16, a Model 890 consists of two card cages (PMB and HP-PB) stacked atop one another. The upper PMB cardcage is further divided into front and back by a backplane. Processor and memory boards are plugged into the front area; additional processor and memory boards can be plugged into the back if necessary.

The lower card cage is an HP-PB bus, which can accommodate the family of HP-PB devices. Device adapters might be half-height or full-height cards, each of which can contain up to four modules. A minimal configuration might consist of a SCSI or HP-FL device adapter for disks and tapes, MUX cards, and data communication boards. Peripherals can be plugged to each board, as with any HP-PB system.

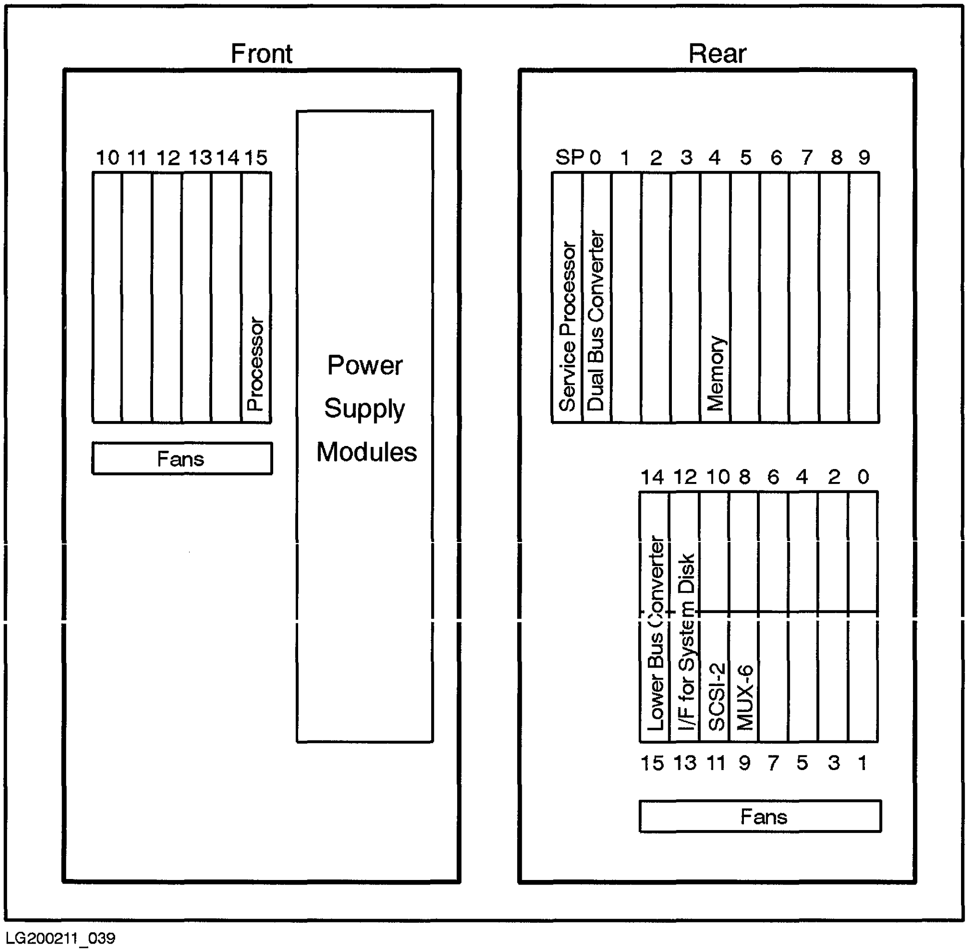


Figure 10-16. Model 890 Card Cage

Figure 10-17 shows the bus architecture of the Model 890.

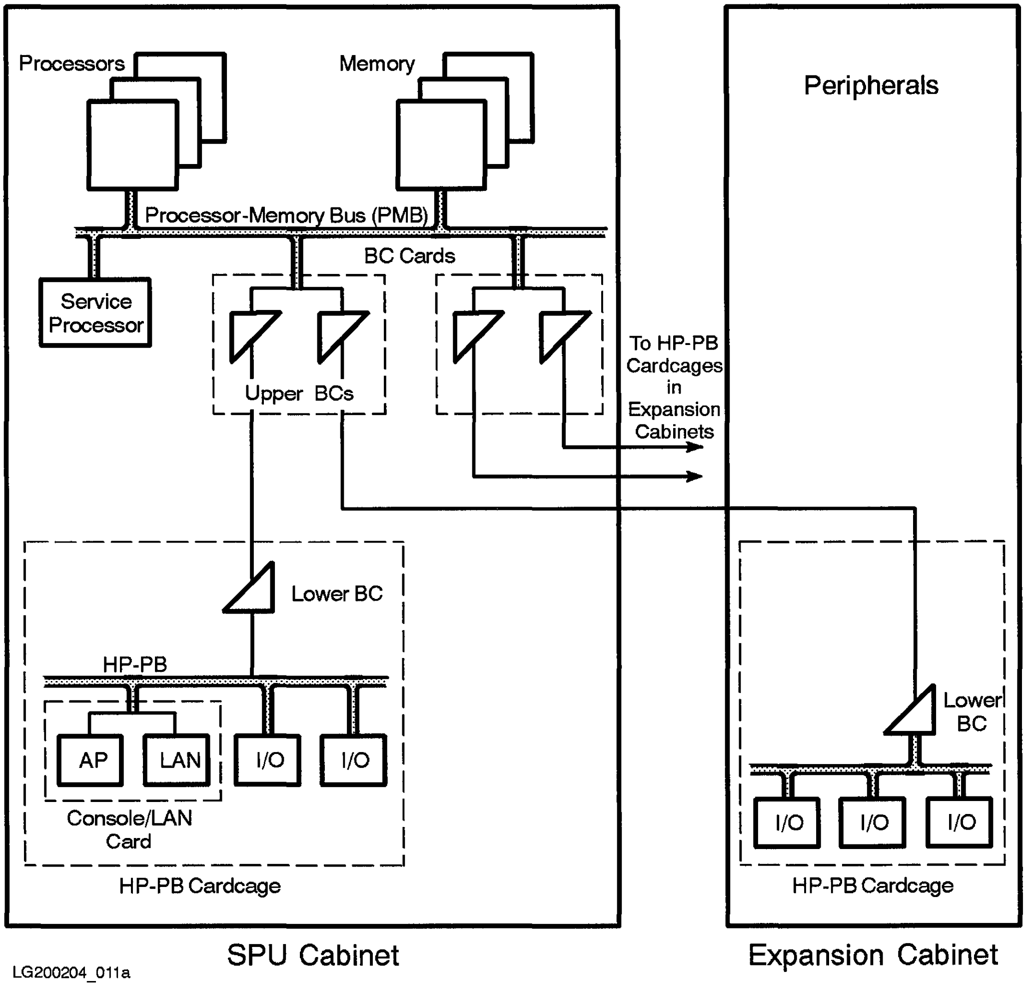


Figure 10-17. Bus Architecture of the Model 890

Communication between the PMB and HP-PB buses is handled by Dual Bus Converter boards, which occupy back slots of the upper cardcage. Model 890 bus converters differ from other systems: two boards perform the conversion from SPU bus to HP-PB bus: a "Dual Bus Converter" board, plugged into the SPU bus for communication with processors and memory, and a "Lower

Bus Converter" board, plugged into the HP-PB bus, for communication with each I/O card cage. As the name implies, each Dual Bus Converter board actually consists of two upper bus converters, which are addressed separately and can provide communication to two Lower Bus Converter boards (that is, two HP-PB card cages). Each bus converter set connects between the PMB and HP-PB buses, using two ribbon cables to accommodate high and low differential voltage.

In addition to the basic HP-PB bus, ribbon cables can connect from Dual Bus Converter boards to external (expander bay) HP-PB card cages.

Addressing on a Model 890

Addressing on a Model 890 is consistent with other HP-PB systems. Bus converters and HP-PB cards are numbered as modules of the slot into which they are plugged (that is, slot times four). Overall, every hardware path for components on the Model 890 conforms to the following convention:

```
bus_converter / HP-PB_card . port_number
```

Because the Dual Bus Converter can provide connectivity from PMB to *two* HP-PB buses, the lower Dual Bus Converter board is numbered:

```
slot number times four
```

The upper Dual Bus Converter boards is numbered with an offset:

```
slot number times four plus two
```

HP-PB cards on the Model 890 are addressed much like Models $8x2$ or $8x7$. Because every slot might accommodate a card having up to four functions, the address of a card in a slot is numbered:

```
slot number times four plus offset
```

The offset is determined by the function and should be documented in the installation guide provided with the card.

Finally, the port number of the HP-PB card is based on a setting on the device itself.

If a SCSI device is attached, a zero is appended to the address. Similarly, HP-FL devices may have an additional number, based on

Model 890 Guidelines

- In powering on the system, the Model 890 requires that you power on all external I/O bays *before* the SPU (PMB cardcage). There are two reasons for this:
  - so that the SPU can find all devices when it configures the system, and
  - to avoid a race condition that would occur if this powering-on order is not observed.
- The Model 890 features new interfaces that can be accessed from various system states. These include:
  - Service Processor (SP). Unique to this system.
  - Processor Dependent Code (PDC).
  - Initial System Loader (ISL).
  - Access Port (AP).

  In addition, new key sequences have been implemented for `TC` (transfer of control) and `RS` (reset).

  For information on these interfaces, see *HP3000 Corporate Business Systems and HP9000 Corporate Business Server Operator's Guide: Series 990/992 and Model 890*, HP part number A1809-90009.

Finding Out About the Current System Configuration

Two tools are available to find out what hardware is configured into your system: /etc/dmesg and /etc/ioscan.

Viewing Hardware Configuration Using /etc/dmesg

Root users can use the command /etc/dmesg to see what is configured on the system. /etc/dmesg displays the contents of an error-message buffer printed by the kernel when it begins to boot-up. While loading, the kernel again checks the buffer for existing hardware. The buffer is filled "FIFO" (first in first out); thus, later messages can overwrite earlier messages.

In the Series 300 and 400, /etc/dmesg shows what cards are present, identified by select code. For example:

```
Initializing hidden mode interfaces
Hidden mode initialization complete
Internal HP-HIL at 0x428000
CONSOLE is ITE
ITE + 1 port(s)
MC68020 processor
MC68881 coprocessor
HP98620C DMA
Internal HP-IB Interface - system controller at select code 7
  Parallel poll interrupts enabled.
HP98644 RS-232C Serial Interface at select code 9
HP98625B High-Speed HP-IB Interface - system controller at select code 14
HP98643 at select code 21
HP98549 Bit Mapped Display at 0x20360000
real mem  = 8376320
using 166 buffers containing 679936 bytes of memory
Root device major is 0, minor is 0xe0000, root site is 0
-- BATTERY BACKED REAL TIME CLOCK
Swap device table:  (start & size given in 512-byte units)
  entry 0 - auto-configured on root device; start = 528444, size = 65772
avail mem = 5177344
lockable mem = 983040
```

In the Series 700, `/etc/dmesg` shows system configuration by select code and function number. It also shows information on the memory subsystem and LAN.

```
Mar 28 15:32
Beginning I/O System Configuration.
Block TLB entry #5 from 0xf9000000 to 0xf9ffffff allocated.
HPA1659A Bit-Mapped Display (revision 8.02/4) in SGC slot 1
SGC at select code 0x10
NCR C700 Core SCSI Interface at select code 0x20 : function number 1
Built in LAN controller found at select code 0x20 : function number 2
HIL interface at select code 0x20 : function number 3
Built-In RS-232C Serial Interface at select code 0x20 : function number 4
Built-In RS-232C Serial Interface at select code 0x20 : function number 5
parallel port at select code 0x20 : function number 6

CONSOLE is on the ITE
I/O System Configuration complete.
Configure called
Root device major is 7, minor is 0x201000, root site is 0
Swap device table:  (start & size given in 512-byte units)
  entry 0 - auto-configured on root device;
start = 1218000, size = 84670
Warning: unable to configure dump device...using primary swap instead.
core image of 4096 pages will be saved at block 634951 on device 0x7201000
@(#)B2352A HP-UX (sys.A.08.05J) #1: Wed Mar 27 23:09:07 MST 1991
physical page size = 4Kb
real mem = 16777216
lockable mem = 7806976
avail mem = 9359360
using 404 buffers containing 3313664 bytes of memory
lan0: AppleTalk node 9
```

In the Series 800, `/etc/dmesg` shows system configuration by hardware path and device driver. It also shows what subsystems and which type of file systems are initialized. For example:

```
Beginning I/O System Configuration.
0 processor
4 cio_ca0
4.0 hpib0
4.0.2 disc1
4.0.3 tape1
4.0.7 disc1
4.1 mux0
4.2 hpfl0
4.2.0 disc2
4.2.1 disc2
4.4 lan0
12 memory
I/O System Configuration complete.
Configure called
Beginning Subsystem Initialization
   nipc initialized
   dskless initialized
   inet initialized
   lan initialized
   uipc initialized
Subsystem Initialization Complete
Beginning Filesystem Initialization
   ufs initialized
   cdfs initialized
   nfs initialized
Filesystem Initialization Complete
Root device major is 10, minor is 0x10d, root site is 0
real mem = 16777216
lockable mem = 7342080
avail mem = 9807872
using 609 buffers containing 3315712 bytes of memory
```

For more information, see *dmesg*(1m) in the *HP-UX Reference*.

Interpreting Hardware Paths on a Series 800

As you can see from Series 800 `/etc/dmesg` output, a hardware path is a numerical sequence of hardware addresses from bus to external device. (The same principle is true for Series 700, although not apparent from its `/etc/dmesg` output.) Data reaches the external device via a software module path—a sequence of programs corresponding to the hardware path and leading to a device driver that controls the external device.

The following table gives sample hardware paths and their corresponding software module paths:

Table 10-3. Sample Hardware Paths

Hardware Path	Software Module Path to Device Driver	Description
16.0	`hpib1.disc1`	an HP-IB disk with bus address 0 attached to slot 4 of a precision bus on a Model 808.
4.4	`cio_ca0.lan0`	a LAN interface card attached to a CIO bus via a channel adapter (`cio_ca0`) on a Model 825.
2/4.0.1	`bus_converter/cio_ca0.hpfl0.disc2`	a disk with bus address 1 attached to an HP-FL interface card plugged into slot 0 of a CIO channel adapter, connected to Mid-Bus slot 1 of a bus converter on a Model 850.

Viewing Hardware Configuration Using ioscan

The command `/etc/ioscan` displays devices on your system by the following major characteristics:

- Hardware path
- Device drivers
- Class of device
- Logical unit number (Series 800 only)

A sample full I/O listing (`ioscan -fu`) of a Model 890 shows the chain of device drivers that correspond to the hardware-path elements:

```
Class         LU  H/W Path  Driver
=============================================================
tty            0  0/28      bus_converter.mux2
disk           6  0/32.0.0  bus_converter.hpfl1.target.disc4
disk           7  0/32.0.1  bus_converter.hpfl1.target.disc4
tty            3  0/40.0    bus_converter.lanmux0.mux4
lan            0  0/40.1    bus_converter.lanmux0.lan3
lantty         0  0/40.2    bus_converter.lanmux0.lantty0
tape_drive     0  0/52.1.0  bus_converter.scsi1.target.tape2
disk           8  0/52.2.0  bus_converter.scsi1.target.disc3
printer        1  0/54      bus_converter.lpr2
disk          12  2/0.0     bus_converter.hpib1.disc1
disk          13  2/0.1     bus_converter.hpib1.disc1
disk          14  2/0.2     bus_converter.hpib1.disc1
disk          15  2/0.3     bus_converter.hpib1.disc1
printer        0  2/1       bus_converter.lpr2
```

Note some characteristic differences among device-driver interfaces, which show up in the `ioscan` output:

- HP-IB device and unit have a `1:1` correspondence.
- SCSI and HP-FL device and units have a `1:many` correspondence. For example, there are two disks in the disk array addressed as `0/32.0.0` and `0/32.0.1`.

`ioscan` has numerous output options; Series 300/400/700 output is more limited than that of Series 800. See *ioscan*(1M) in the *HP-UX Reference* for full specifications.

11

System Configuration

What Does Configuring Mean?

When you configure your system, you inform the operating system what hardware is present by associating the hardware with necessary software—device drivers and device files. You might also have to set values or limits—configurable operating-system parameters—for optimal system performance.

Configuring your system is important when you initially install it, when you make certain major changes to HP-UX, or when you add a device and the driver is not present in the currently running kernel. For example, when you add a printer, not only must the peripheral be physically attached, but the executable code (the device driver, such as `lpr1`) and the printer device file (such as `/dev/printer`) must also be present. Or, a new application might need more data space, which would require adjusting parameters.

Once the necessary files are created or included in the configuration file, you must regenerate the HP-UX kernel to activate the change. Much configuration occurs automatically, when you reboot the system. Sometimes, however, you must override the defaults or attach a peripheral device that cannot be configured automatically. For those instances, you use SAM menus (or HP-UX commands) and reboot the system to enable the configuration.

This chapter describes the software elements used by the operating system to activate your system configuration.

Understanding the Role of the Kernel

The **kernel** is the software core of HP-UX and is the part of the operating system that deals directly with the hardware. The kernel also manages the low-level functions for the operating system and insulates the operating system from the details of the hardware.

As a system administrator, you identify devices on a hardware level by their physical address. The kernel recognizes any hardware by two elements—device drivers and information found in the device files. For the kernel to find and use any class of hardware, the appropriate device drivers must be included in the kernel and the appropriate device files must be installed in the file system.

Figure 11-1 shows the kernel in relationship to the entire HP-UX environment:

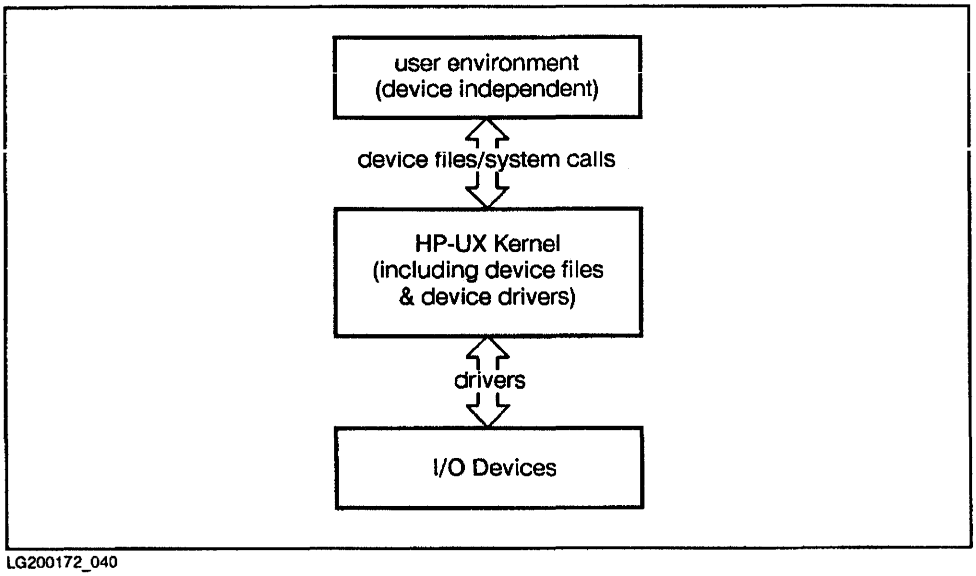


Figure 11-1. Kernel in Relation to the User Environment and I/O Devices.

Kernel Configuration Files

The `dfile` for Series 300/400/700 or `S800` for Series 800 is the principal vehicle from which the kernel derives its image of your system. `dfile` and `S800` are termed **kernel configuration files**. Although they differ in name and format, `dfile` and `S800` are both used to create the kernel for their respective systems; both contain information vital to the kernel about device drivers, subsystems, swap configuration, and configurable parameters.

Every time you change the kernel configuration file, you have to generate a new kernel and reboot the system to activate the changes in that file. Figure 11-2 compares what happens when you generate a new kernel on Series 300/400/700 and Series 800.

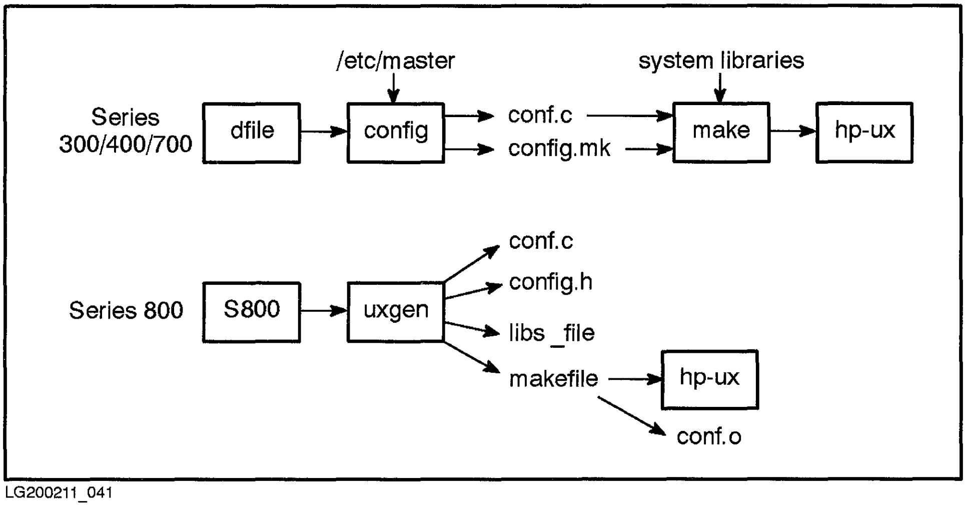


Figure 11-2. Kernel Configuration Files.

dfile—Series 300/400/700 Kernel Configuration File

The `dfile` is the Series 300/400/700 configuration description file, used to create a new or customized kernel with either *config*(1M) or SAM. By convention, `dfile` is located in the `/etc/conf` directory. Several templates reside in `/etc/conf` for both server and client configurations. Use the `dfile` to specify configurable parameters and device drivers.

The `dfile` shown on the next page is written for both bit-mapped and remote terminal consoles, and includes the `cs80`, `98624`, `98625`, `98628`, and `98642` drivers. It also specifies the default kernel for a diskless cluster node being added to an HP-UX cluster, and is the template used by SAM if you specify the option "Create a minimally loaded operating system."

The *config*(1M) manual page in the *HP-UX Reference* provides detailed information on the component parts of the `dfile` and how to customize it for your own system needs.

Note that the notation in the `dfile` beginning with an asterisk (*) or `/*` are commentary, ignored by the kernel generation process.

For explicit procedures for generating a new kernel, refer to the *System Administration Tasks* manual.

Here is a sample `dfile` for the Series 300/400:

```
* DEVICE DRIVERS
cs80                        /* disk and tape drivers */
scsi
tape
stape

printer                     /* printer drivers */
* pseudo terminal drivers
ptymas
ptyslv

srm                         /* shared resource management driver */
rje                         /* remote job entry */
dskless                     /* Diskless code pseudo-driver */

* Network File System (NFS) and networking drivers
nfs
uipc
nipc
inet
netman
lla
lan01
ni
netdiag1
*cdfs                        /* CD File System (CDFS)  */
meas_sys                     / * Measurement System driver */

* CARDS
98624                        /* HP-IB interface  */
98625                        /* high-speed HP-IB interface */
98626                        /* RS-232C serial interface */
98628                        /* RS-232C datacomm interface */
98642                        /* RS-232C multiplexer */
98658                        /* DIO SCSI interface */
98265                        /* SCSI interface */

* SWAP CONFIGURATION
* CONFIGURABLE PARAMETERS
dskless_node 1
ndilbuffers  1
npty   30
* Disable support for HP98248A (Floating Point Accelerator)
* by changing the 1 to a 0 in the following line.
fpa 1
```

Here is a sample `dfile` for a Series 700 system:

```
* Installed software drivers and I/O interface cards
autox                           / * Magneto-Optical autochanger */
autoch

* SCSI disk and DAT device drivers
scsi
scsitape
scsifloppy
sim0

parallel                        /* Parallel interface driver */

* Network management
netman
nipc
uipc
inet
lan01
asio0
nfs                             /* Network File System (NFS) */
cdfs                            /* CD File System (CDFS) */

* PTY pseudo drivers
pty0
pty1

dconfig                         /* libIO Kernel interface */
dmem                            /* Memory Diagnostics  */

dskless                         /* Diskless cluster code pseudo driver */
rdu                             /* Remote Swap for diskless */

* I/O drivers
eisa
CharDrv
hpib
cs80
hshpib

* Tunable parameters
dskless_node 1
server_node 1

* Swap info
*swap auto              /* Swap after fs on root device -- default. */
*swap scsi 201500 -1    /* Swap after fs. */
*swap scsi 201400 0     /* Use the whole disk for swap. */
```

S800 file—Series 800 Kernel Configuration File

The S800 file (typically, `/etc/conf/gen/S800`) is used by *uxgen*(1M) or SAM to generate a new kernel for Series 800 computers. Templates for each model reside in `/etc/conf/gen/templates`, with executable scripts used by `regen` to customize a configuration file appropriate to your system. The S800 file, structured differently than the `dfile`, contains:

- An include statement for the `/etc/master` file, which the kernel uses for pointers to modules, device-driver implementation, subsystems, and default operating-system parameters.
- Device drivers that govern I/O for subsystems and peripherals.
- Operating-system parameter values (see Appendix A of *System Administration Tasks* manual)
- Addresses of kernel devices (such as console and swap), which can be specified as default, as sections, or as logical volumes. (The S800 example on the next page shows an LVM implementation. a standard implementation might show kernel devices as follows:

```
console on default;
root    on default section 10;
swap    on default section 1;
dumps   on default section 1;
```

- `io` statement, used to configure devices that cannot be bound automatically, overrides the system's automatic configuration capabilities. (The `io` statement is discussed in more detail later in this chapter.)

Refer to *uxgen*(1M) in the *HP-UX Reference* for information on the component parts of the S800 file and how to customize it for your own system needs. See *System Administration Tasks* manual for procedures on generating a kernel.

Here is a sample S800 file on a Model 837 running LVM:

```
#include "/etc/master"
include mirror;              /* subsystem and driver include statements */
include switch;
include nipc;
include lv;
include lvm;
include nfs;
include lan;
include ni;
include nm;
include inet;
include uipc;
include target;
include tape2;
include scsi1;
include mux2;
include memory;
include lpr2;
include lan3;
include disc3;
nclude hpib1;

timezone 480;        /* sets system clock to minutes from GMT */

console on mux2 at 56;    /* kernel devices */
root on lvol;
dumps  on lvol;
swap on lvol;

io {}                 /* io statement */
```

The io Statement of the S800 File

Series 800 systems automatically bind device drivers included in the S800 file to their appropriate device. This "autoconfiguration" applies to most device drivers. You must use the io statement to configure drivers that cannot be bound automatically into the kernel using SAM. This section describes the io statement of the S800 file.

The syntax of the io statement is:

```
io {
  address_specification
       :
}
```

address_specification is a set of one or more nested device-driver and address specifications for the installed non-autoconfigurable buses, bus converters, channel adapters, I/O cards, and peripheral devices.

The io statement contains one or more nested *address_specification* for each level of:

- Bus converter (for example, plugged into SMB and Mid-Bus)
- Channel adapter (plugged into the Mid-Bus on CIO machines or optionally into the HP-PB bus)
- Device adapter manager
- Device manager

Hardware paths can be derived from information in the io statement of the S800 file:

```
    bus_converter address 2 {
 cio_ca0 address 4 {
   hpib0 address 2 {
      instr0 address 0;
```

In this example, the hardware path for an instr0 device is 2/4.2.0, derived by reading the number following "address" in each line of the nested io statement leading to and including instr0.

The io Statement of a Mid-Bus/CIO Configuration.

Suppose you have a Model 835 containing:

- two HP-IB disk interface cards, each with four disks connected
- two MUX cards
- another HP-IB interface card with:
  - two line printers
  - two tape drives
  - a plotter
- a LAN card
- a GPIO card

Since most devices listed are configured automatically, their device drivers are referenced as include statements.

```
include cio_ca0;
include hpib0;
include disc1;
include mux0;
include lpr0;
include tape1;
include instr0;
include lan0;
include gpio0;
```

Two devices listed—plotter and GPIO card—cannot be configured automatically (see Table 11-1), and therefore must be specified in the `io` statement, by device driver and address. The device drivers are defined in `/etc/master`; to determine which driver to use, consult the *Installing Peripherals*. The address is based on the module number of the channel adapter board on the system bus or the slot number of the device adapter board.

Here is an `io` statement for the plotter and GPIO card:

```
io {
  cio_ca0 address  4 {            address of CIO channel adapter
    hpib0 address  2 {            address of HP-IB card for tapes & printers
      instr0   address  7;        address of plotter
    }
    gpio0  address  5;            address of GPIO card
  }
}
```

The first level of nesting in this `io` statement shows the CIO channel adapter, which is used by both devices to communicate with the system bus and processor:

```
cio_ca0 address 4 {
   address_specifications
   :
}
```

`cio_ca0` The name of the controlling CAM (channel adapter manager, explained in the section, "Device Drivers").

`address 4` Physical location of the channel adapter module in slot 1 of the Mid-Bus. Module boards may contain multiple modules; the address is set as slot number multiplied by 4 so that other modules on the board can be addressed by offsets.

The addresses specified after the CIO channel adapter include all device adapters inserted in the CIO Bus card cage that cannot be configured automatically. The same pattern of driver name and address is used:

`hpib0` The device adapter manager that controls the HP-IB card.

`address 2` The physical slot of the HP-IB card in the CIO-Bus card cage.

`instr0` The device driver that controls the plotter. Because the plotter is attached to the HP-IB card, `instr0` is specified within the nested address of `hpib0`.

`address 7` The bus address of the plotter card (in this case, an HP-IB address switch located on the plotter).

`gpio0` The device driver that controls the general-purpose I/O (GPIO) card.

`address 5` The physical slot of the GPIO card in the CIO-Bus card cage.

The io Statement of an HP-PB Configuration.

To configure a plotter and GPIO card to a Model 857, your `io` statement might look as follows:

```
io{
 hpib1 address 44{
  instr0 address 31;
  }
 gpio1 address 48;
}
```

As in the CIO example, the HP-PB `io` statement consists of a nesting of device drivers and addresses for each level of architecture, from the Precision Bus to the peripheral device.

The module address `44` locates the HP-IB board at slot number 11 of the Precision Bus; module address `48` locates the GPIO board at slot 12. As in the CIO example, the plotter uses the `instr0` device driver. The plotter's HP-IB address is set to `31`.

When Do You Reconfigure the Kernel?

For most purposes, the kernel generated when you bring up the system will serve satisfactorily. However, some circumstances might require you to modify the kernel.

Simple changes are performed most easily using the SAM utility. (See the *System Administration Tasks* manuals for your specific system for instructions on modifying the system.)

The following tasks require kernel configuration:

- Change the configuration of I/O cards and device drivers.
- Add or remove a type of peripheral device, such as a disk, tape drive, or CD-ROM.
- Improve system performance.
- Change the swap configuration.
- Add or remove optional subsystems such as NFS, LAN, FDDI, or diagnostics.
- Remove unused I/O drivers or subsystems to reduce kernel size.
- Modify tunable operating-system parameters.

To accomplish these tasks, you modify the kernel to include a necessary device driver or subsystem, or change configurable operating-system parameters. Modifying the kernel involves changing the `dfile` or `S800` file. The next sections will examine the kinds of things you change when you reconfigure the kernel.

Statements to Include Device Drivers

In both `dfile` and `S800` file, you include statements for device drivers. In the `dfile`, device drivers are simply listed. In the `S800` file, the device drivers are listed as `include` statements. Each listing corresponds to a device driver for subsystem, I/O card, or peripheral device found on the system that can be automatically configured. The kernel finds the source code for all included device drivers by referencing through the `/etc/master` file. (The `/etc/master` file is discussed later in this chapter.)

Kernel Devices

In the Series 300/400/700, kernel devices are found via the `/etc/master` file.

In the Series 800, kernel devices—`console`, `root`, `swap`, and `dumps`—are cited separately in the `S800` file to allow the user to modify them. Whenever possible, they are listed with default hardware addresses. Defaults are derived from processor-dependent code, as follows:

- The default console path is the address displayed during the boot process, before you get an `ISL` prompt.
- The default for the root device is the place from which the kernel was loaded, usually indicated by the primary boot path. Normally, the root disk defaults to section 13 of the disk from which you loaded the kernel. If you are using the Logical Volume Manager (LVM), your root logical volume is in your root volume group. (For information on implementing LVM, see Chapter 9, "Logical Volume Manager" in this manual and "Managing Logical Volumes" in *System Administration Tasks* manual.)
- Swap defaults to section 15 of the root disk or a logical volume in the root volume group, if you are implementing LVM.
- The dump device defaults to primary swap.

Defaults can be changed by executing the `hpux` command with options at the `ISL` prompt during installation. See *hpuxboot*(1M) in the *HP-UX Reference* and *Installing and Updating HP-UX*.

Kernel Dump Device

Your HP-UX system is very robust; however, certain situations (such as data segmentation fault or illegal instructions) can cause the system to panic. And as the system goes down, the kernel writes (dumps) an image of the memory core to the dump device(s) and reboots. (If the system hangs, you can generate a core dump from the console, by entering "console mode" and using (control-B) and the `tc`, "Transfer of Control" command.)

During the reboot, the `savecore` code is executed by `/etc/rc`. `savecore` locates the core image and deposits a `core` file in the file system (typically in `/tmp/syscore`). Once reboot is completed, the `core` file can be retrieved and a programmer can debug the program using standard debugging tools, such as *adb*(1).

Typically, core dumps write to primary swap. On workstations, this is true by default; on Series 800 systems the `S800` file designation `dumps on default` causes core dumps be written to primary swap.

This default is adequate, provided primary swap is large enough to accommodate a "snapshot" of the entire real memory. If primary swap is smaller than real memory and no other dump device is designated, only a partial core image is saved when a system crashes.

If your system is configured with memory that exceeds the size of primary swap, you can configure other dump options to ensure that a total core image is saved if the system crashes.

On Series 300/400/700, you can use entire devices to share `swap` and `dump` purposes. Use same syntax for `dump` as you use optionally for designating `swap` in the `dfile`. Note, do *not* have a file system on this device, because there is a small chance you might lose data. Also, you *must* have `swap` on the root device, since `/hp-ux` requires it to boot.

On the Series 800, you can specify multiple `dumps` devices in the `S800` file. These devices can supplement primary swap or substitute for primary swap. If supplementing primary swap, the primary swap device *must* be the first `dumps` device listed, or `uxgen` will fail. Both `swaps` and `dumps` devices can be specified as disk sections, logical volumes, or even dedicated disks. On Series 800, the `dumps` designation in the `S800` file tells you the location of the `core` file.

For complete information about setting up **dumps** areas, consult Chapter 7, "Managing Swap Space" of the *System Administration Tasks* manual for your system. Also see *savecore*(1M), *config*(1M), *swapon*(1M), and for systems implementing LVM, *lvlnboot*(1M), in the *HP-UX Reference Manual*.

Operating-System Parameters

Configurable operating-system parameters are values set in the `dfile` or `S800` file to customize kernel characteristics. Knowledgeably setting them can optimize performance, but proceed cautiously, because many parameters are interrelated and imprudent changes might cause unpredictable results!

Operating-system parameters affect the following areas:

- inter-process communication
- swap space allocation
- memory management
- cluster system management
- process memory limits
- message mapping, queueing, and size characteristics
- file system buffer cache
- networking process scheduling and memory allocation
- maximum number of system-wide open files, file locks, text descriptors

Operating-system parameters and their dependencies are listed in Appendix A of the *System Administration Tasks* manual.

What Happens when you Regenerate the Kernel?

Series 300/400/700 systems execute *config*(1M) using information in `dfile`; Series 800 systems execute *uxgen*(1M) using information in `S800`.

For HP-UX to use an I/O device, the device must be physically connected and turned on. As a system administrator, you identify I/O devices by their physical address, but the kernel identifies them by device driver and device files. Then, to actually communicate with the I/O device, certain logical (software) associations must exist, including device driver, hardware address, logical unit number, and device file.

Kernel Regeneration—the Series 800 Case

During the boot process, the kernel reads the `S800` file to include device drivers for all hardware components, to mount the root device, to open swap devices, and to establish the dump device(s). The kernel launches `init`, the initialization process, which reads `/etc/inittab`, performs any required machine-dependent initialization, spawns processes, and calls `ioinit`. `ioinit` reads a reference file called `/etc/ioconfig`, which maintains mapping information of device driver and logical unit number to the hardware addresses of devices. Figure 11-3 shows this sequence of events.

As system administrator, you cannot directly access `/etc/ioconfig`, whcih is maintained by the kernel. When you change the system configuration and reboot, the system probes the hardware for all devices attached, and maps the information into `/etc/ioconfig`.

Changing your system's configuration causes the kernel to probe the system twice: The first time, the kernel scans the backplane, identifies all cards it finds, initializes global interrupt-type counts, and determines the size of the physical address space to be mapped into virtual space. Data structures inside the kernel probe all hardware to determine what is hooked up (configured) into the system.

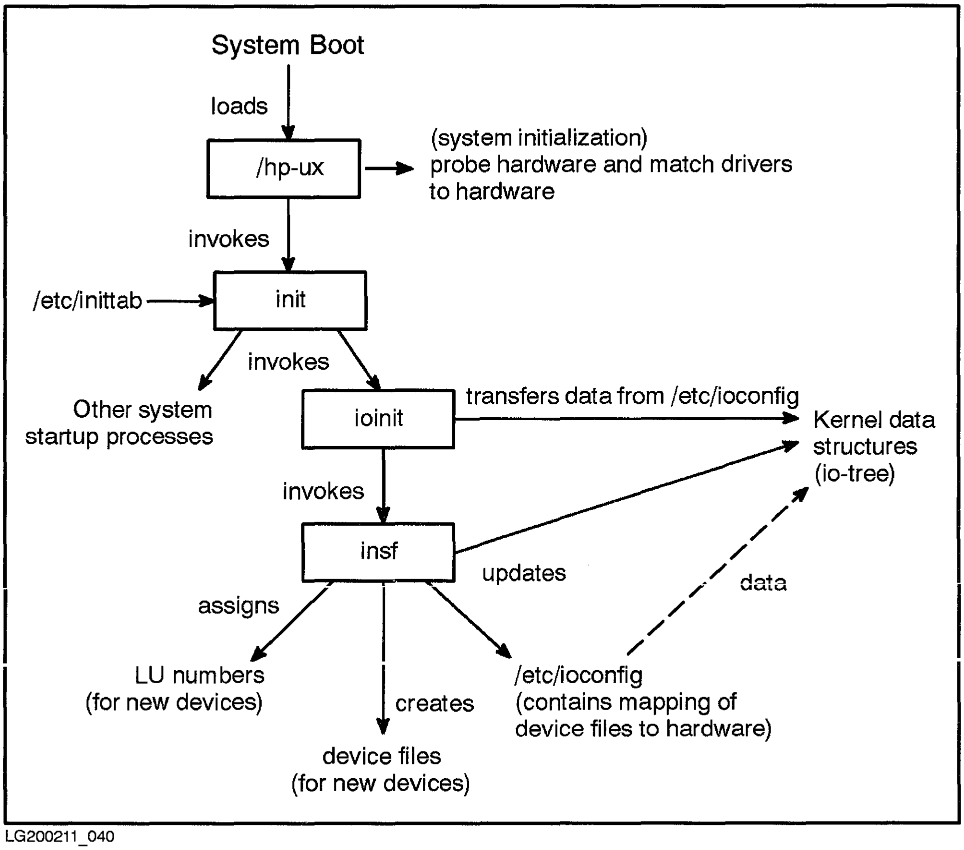


Figure 11-3. System Initialization by the Kernel

The kernel detects all buses, channel adapters, device adapters, and external devices and binds all hardware address space pointers to their corresponding software modules (device drivers) and device special files. This is done by writing to the kernel's internal data structure, `io_tree`, as well as to `/etc/ioconfig`. Then, `ioinit` calls `insf` to assign logical unit numbers to all newly configured devices and downloads the `lu` numbers into the kernel. `insf` then creates device files for any new devices it finds, then updates `/etc/ioconfig`.

Having completed the I/O mapping, `init` then creates a process for each terminal on the system, thus establishing a multi-user environment.

Figure 11-4 illustrates the auto-configuration process and shows user access to the configuration after the boot process.

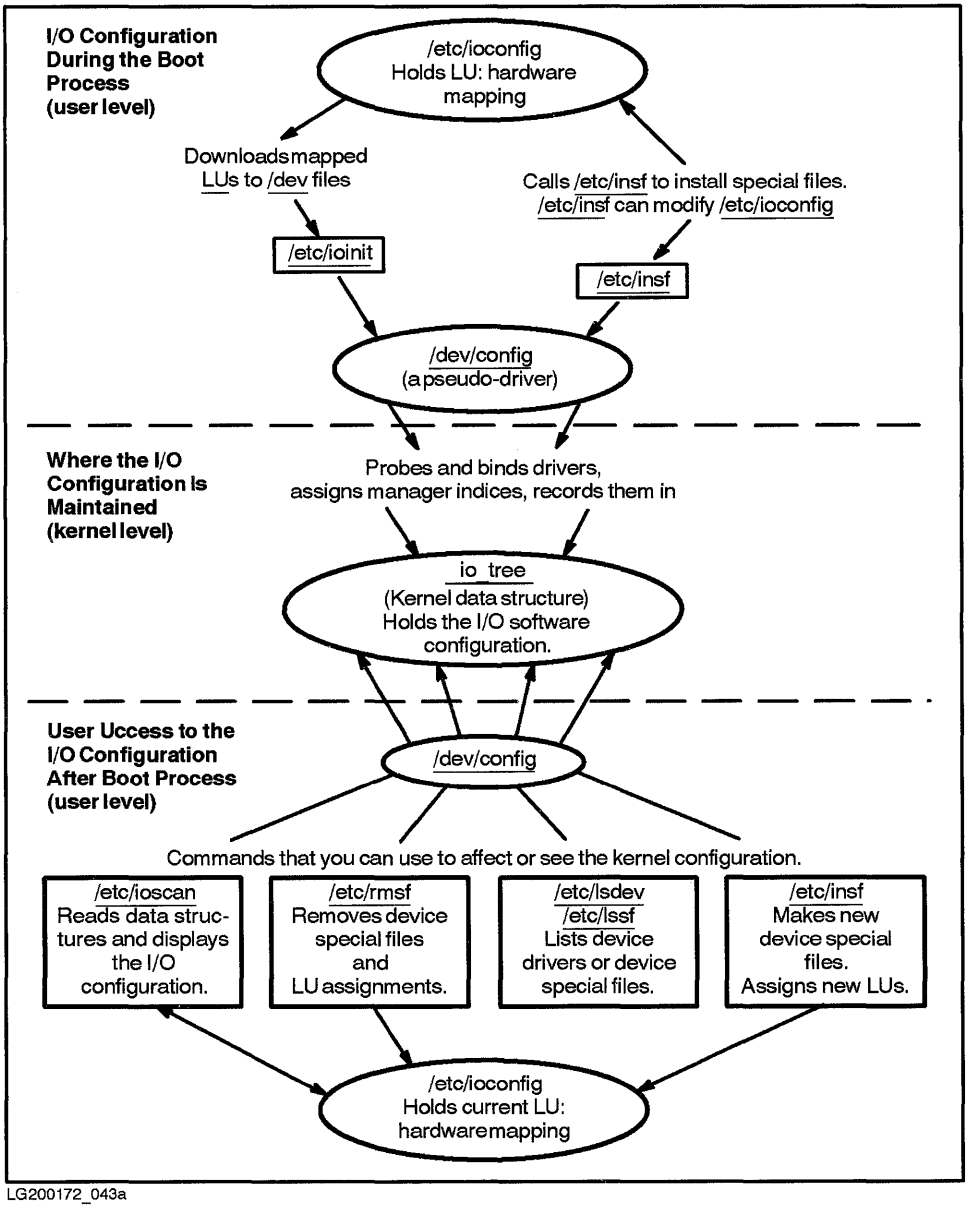


Figure 11-4. Series 800 I/O Auto-configuration

Device Drivers

Compiled inside the kernel for all HP-UX systems are software modules known as **device drivers** that control I/O for a particular device or class of devices. All drivers shipped are by convention located in a kernel archive library directory, `/etc/conf/lib`.

The kernel identifies the device drivers it needs via its kernel configuration file and finds the device driver via the `/etc/master` file.

Although the term is used generically throughout this text, "device driver" actually summarizes a hierarchy of programs known collectively as I/O managers. I/O managers control the various levels of hardware in the bus architecture. Note evidence of this hierarchy in both the previous `S800` file (the indented lines in the `io` statement, discussed earlier in this chapter).

Each kind of I/O manager manages a specific level (or levels) of architecture. In the Series 300/400/700:

Interface driver
: Controls the interface card for handling memory-mapped and processor I/O.

Device driver
: Controls the device for kernel data structures to communicate from the interface card to the external device itself.

In the Series 800, I/O managers are somewhat more complex, but in the simplest case, might be defined as follows:

CAM
: The channel adapter manager manipulates data between the Mid-Bus and CIO buses; for example, `cio_ca0`.

DAM
: The device adapter manager controls the communication between the I/O bus (such as CIO) and interface card. For example, `hpib0` controls HP-IB data protocol transfer for the CIO bus and `hpib1` controls HP-IB data protocol transfer for the HP-PB bus.

DM
: The device manager controls device-specific communication; for example, `lpr10` for a ciper printer.

LDM
: The logical device manager provides the software interface between the user and the operating system, is used as an interface to a class of DM, and handles generic requests (such as buffering and queueing); for example, `lpr0`.

Generally, each I/O manager corresponds to a layer of hardware, but many Series 800 drivers are written to encompass more than one layer. These are known as monolithic drivers. For example, `mux0` is a monolithic driver that combines LDM, DM, and DAM, and transfers requests between the HP-UX kernel and the channel adapter manager, `cio_ca0`.

In the Series 800, each device driver (DM and/or LDM) corresponds to a class of hardware, such as `disc1` for all HP-IB disks, `disc2` for HP-FL disks connected through CIO, `disc3` for all SCSI disks in a CIO or HP-PB system, or `disc4` for HP-FL disks in an HP-PB system.

The device driver enables the kernel to communicate with the device. The kernel communicates with the hardware by associating the device-driver name with a hardware address. The kernel derives its information about device drivers and their corresponding hardware addresses from the `dfile` for Series 300/400/700. On Series 800 systems, the kernel scans the hardware to derive information about device drivers and their corresponding hardware addresses, as shown in Figure 11-2.

In the Series 300/400/700, device drivers are listed in the `dfile` and the `config` program derives device-driver definitions from the `/etc/master` file.

In the Series 800, device drivers are specified in several areas of the `S800` file: `/etc/master`, `include` statements, and `io` statement.

The /etc/master File

All HP-UX computers use a `/etc/master` file, which contains critical device driver definitions in terms of file type (block or character), major number, and I/O functions. These definitions are used by *config*(1m) (on Series 300, 400, and 700) and *uxgen*(1m) (on Series 800).

On the Series 300/400/700, the `/etc/master` file consists of lists of devices and cards, their associated interface or device driver, major numbers by block or character type. On the Series 700, the `/etc/master` file also lists the subsystems for which library a given device driver is dependent.

On the Series 800, the `/etc/master` file defines device drivers in a format resembling C syntax. The first entry in the `S800` file includes the `/etc/master` file.

Because the bus architectures of Series 800 computers are hierarchical, the `/etc/master` file also reflects the hierarchical I/O manager relationships of parent and driver. For example, the following excerpt identifies `cio_ca0` as the parent driver for the `hpfl0`, the device driver that controls a CIO fiber-link card:

```
module hpfl0 {          /* CIO HP-FL card 27111A */
    parent is cio_ca0;
 driver hpfl0;
 }
```

Note Since `/etc/master` contains critical information used by *config*(1M) (for Series 300/400/700) and *uxgen*(1M) (for Series 800), do not alter it.

Device Driver Configuration

You add or delete device drivers when installing or updating the system, or adding a peripheral device to a system that lacks the associated device driver. Use the procedures in *System Administration Tasks* manual and *Installing Peripherals* manuals for your particular system.

When configuring a new device into the system, Series 300/400/700 and 800 perform somewhat differently.

- In the Series 300/400/700, you invoke SAM, reconfigure the kernel, and reboot the system to inform the kernel of all configured devices.

  On Series 400 and 700 systems equipped with an EISA bus, when the system boots, the `eisa_config` utility runs in automatic mode to ensure that all resources on the EISA bus are properly provided for.

  However, kernel drivers `cs80` (Series 300/400/700) and `hshpib` (Series 700 only) are not included in the default configuration. These drivers are needed for the optional EISA HP-IB card and its peripherals (for example, to communicate with an HP-IB DDS-format drive). To configure them, you must invoke SAM to add the `cs80` device driver, which also configures `hshpib`, a dependent driver. When the system is rebooted, SAM writes the drivers to the `dfile.sam` file. If you build the kernel manually, you must add the device drivers to your `dfile`. To view a list of configured device drivers and their major numbers, run `lsdev` (see "Major Numbers and lsdev" later in this chapter).

- In the Series 800, if the necessary driver is in the kernel, you do not have to reconfigure the kernel. Using `ioscan` and `insf`, the system locates the devices and binds their necessary device driver. (You can run `ioscan` and `insf` manually; if you reboot, the system runs them automatically.) If the driver is not in the kernel, use SAM to add the driver or to configure a device that is not automatically configurable. For the few drivers not automatically configurable even by SAM, you must edit the `io` statement of the `S800` file, then regenerate the kernel. When you reboot, the system finds the devices and binds their necessary device drivers.

Table 11-1 lists device drivers and whether they can be automatically bound into the Series 800 kernel. (Note that the driver must be added, usually by SAM, if it is not included in the S800 file.)

Table 11-1. Device-Driver Capabilities (Series 800)

Device Driver	Description	Bound Automatically?
autox0	Cartridge tape management	yes
cent0	parallel SCSI interface	yes
cio_ca0	Channel adapter	yes
disc1, disc2	Disk-drive management	yes
disc3, disc4	Disk-drive management	yes
display0, graph0, graph2	Graphics management	yes
fddi	FDDI networking protocol	yes
gpio0	AFI (Asynchronous FIFO Interface)	no
gpio1	General-purpose parallel interface	no
hpfl0, hpfl1	Fiber-link interface	yes
hpib0, hpib1	HP-IB devices	yes
instr0	Plotter and HP-IB instrumentation	no
lan0, lan1, lan3	LAN (networking)	yes
lpr0, lpr1, lpr2	serial and parallel printers	yes
mux0, mux1, mux4	terminals, consoles, keyboards	yes
osi0	LAN card for MAP networking	yes
psi0	Programmable serial interface (networking)	no
pdn0	Public data network	no
rti0	Real-time interface	no
scc1, lantty0	terminals, consoles, keyboards	yes
scsi1, scsi2	SCSI-interface management	yes
tape1, tape2	tape drives	yes
token2	Token Ring IEEE 802.5 (networking)	yes

Pseudo-Drivers

Pseudo-drivers are another kind of device driver found in the configuration description file.

A pseudo-driver (such as `/dev/config`) is a piece of code in the kernel, like any other driver, but not actually associated with a device. Instead, pseudo-drivers are used by other logical device files such as `/dev/null` (the "bit bucket") and `/dev/kmem` (kernel memory) and enable them to open, close, read, write, and perform I/O control.

A `pty`, meaning pseudo-terminal, is another example of a pseudo-driver. The `pty` has two elements—master and slave—both of which are necessary for emulating a terminal's functions. These elements are used by programs that must address another piece of code capable only of communicating with a terminal, such as a test program that interacts with `vi`.

Device Files

HP-UX communicates with peripherals through files called **device files**, which are kept in `/dev` or one of its subdirectories. Device special files contain no actual data, but exist solely as the way to write to or read from a device. Device files point to the logical address for peripherals and other devices.

Device files contain information encoded in major and minor numbers to identify device drivers, hardware addresses, and other characteristics. HP-UX uses this device-driver and address information to manage all data transfers to and from the device.

In the Series 300/400/700, the device-file name encodes hardware-address information directly. In Series 800, the hardware information must be inferred from the minor numbers, found either by running `ll` on the device file or (for Series 800 only) by running the `/etc/lssf` or the `/etc/ioscan` command.

Device files link devices to the kernel and enable I/O between the kernel and devices. HP-UX uses device files to determine which driver to use when communicating with a device and the device's hardware address. (Valid drivers must be compiled into the running kernel.)

The kernel treats I/O to a device much like I/O to a file, by performing I/O operations as if through the file system. For example, each terminal has its own device file through which HP-UX writes data (which appears on the terminal screen) and reads data (typed by the user at the terminal keyboard).

Typically, two device files—a block and a character device file—are created for each I/O device in the system. (These are described shortly.)

Individual device files can be created by the following commands:

- *insf*(1m)—Series 800 only.
- *mksf*(1m)—Series 800 only.
- *mknod*(1m)—Series 300/400/700/800.

On the Series 800, `insf` is the easiest, fastest way to handle special files. `insf` can be used any time. It is not actually run when the kernel is regenerated, but rather when the new kernel is booted. `insf` assigns logical unit numbers, needed for the device before you can use either `mksf` or `mknod` to create a device file. `mksf` and `mknod` should be used only for special cases, as they are more difficult to use than `insf`.

Characteristics of Device Files

As a system administrator, you need to understand the following characteristics of device files to interpret output from a system configuration or to configure devices on a system:

- device file directories
- I/O type (block and character device files)
- device file listings
- device file naming conventions
- major and minor numbers

Organization of Device File Directories

All device files should be located in the `/dev` directory. Many commands expect to find device files in `/dev` and fail if the required device file is not there. Peripherals are associated with block and character (raw) special files found in the `/dev` subdirectories, as listed in the following table:

Type of peripheral	Block Mode	Character Mode
Disk drives	`dsk`	`rdsk`
Cartridge-tape drives	`ct`	`rct`
Magnetic and DDS-format tape drives	`mt`	`rmt`
Graphics framebuffer devices	`crt`	
Floppy disk drives	`floppy`	`rfloppy`
Autochangers	`ac`	`rac`

Device File Directories in a Cluster (Series 300/400/700 only)

If your system is an HP-UX cluster, the `/dev` directory is a context-dependent file (CDF).

Log into the computer to which the peripheral is attached when adding or changing device information in a cluster. See *Managing Clusters of HP 9000 Computers: Sharing the HP-UX File System*.

Block versus Character Device Files

There are two main types of device files: block and character, represented by the characters `b` and `c`, respectively. All I/O devices can be classified as block or character.

Block device files transfer data using the system buffers to speed up I/O transfer.

Storage devices such as tape drives, hard and floppy disks, and magneto-optic drives can use block device files.

Character device files transfer data blocks of varying sizes one character at a time without using the system's I/O buffers. The user program does its own buffering. Typically, the following devices use character device files: terminals, printers, plotters, digitizers, magnetic tape drives, cartridge tape drives, and disk mass storage devices. Character I/O transfer is also called **raw I/O transfer**, and sometimes character devices are called **raw devices**.

Disk drives generally need both block and character device files. The file system accesses ordinary files and directories with block device files, using the buffer cache for speed. A database can use a character device file to access an area of the disk that the file system does not use. File system integrity is checked using `fsck` through the character device file, since `fsck` requires low-level control of the device.

See the *HP-UX Reference* for manual page specifications on `fsck` and `newfs`.

Device File Listing Format

When you execute an `ll` command from the `/dev` directories, you see a device-file entry with different information than other long file listings:

```
brw-rw-rw-  1 bin   bin   0 0x400222 Apr 5 1990 /dev/ct/c2d1s2
```

The first character in the first field identifies the file type; that is, whether it is a character (`c`) or block (`b`) device. (Remaining information in the first and next three fields resemble any file listing, with file access permissions, linkage, owner, and group.)

Then, instead of a field showing the size of a file, device files feature two unique fields: major number and minor number (in this case `0` and `0x400222`). Major and minor numbers derive from hardware-path information given to commands for creating the device file. (See "Overview of Major and Minor Numbers" for further discussion.)

If the file is a symbolic link, you will derive more accurate output by using `ll -L`, which lists the file or directory to which the link refers.

In the Series 800, detailed information about a device file can be displayed by using the `/etc/lssf` command. For each device file specified, `lssf` displays the major number, file type, characteristics encoded into the minor number, and logical unit specification (if present).

`lssf` can be thought of as a corollary to *mksf*(1m). For example, a device file created with the command

```
mksf -d disc2 -l 1 -u 0 -s 3 /dev/dsk/c1d0s3
```

would be decoded:

```
# lssf /dev/dsk/c1d0s3
disc2 lu 1 unit 0 section 3 address 4.2.0 /dev/dsk/c1d0s3
```

Another tool, *ioscan*(1M), displays the following output for the same information:

```
# ioscan -f /dev/dsk/c1d0s3

Class  lu  H/W Path  Driver              H/W Status    S/W Status
=================================================================
disk    1  4.2.0    cio_ca0.hpfl0.disc2  ok(0x2)       ok
```

See *lssf*(1M), *mksf*(1M), and *ioscan*(1M) in the *HP-UX Reference*.

Overview of Major and Minor Numbers

The kernel recognizes a device and its device driver by numbers associated with device files called major and minor numbers.

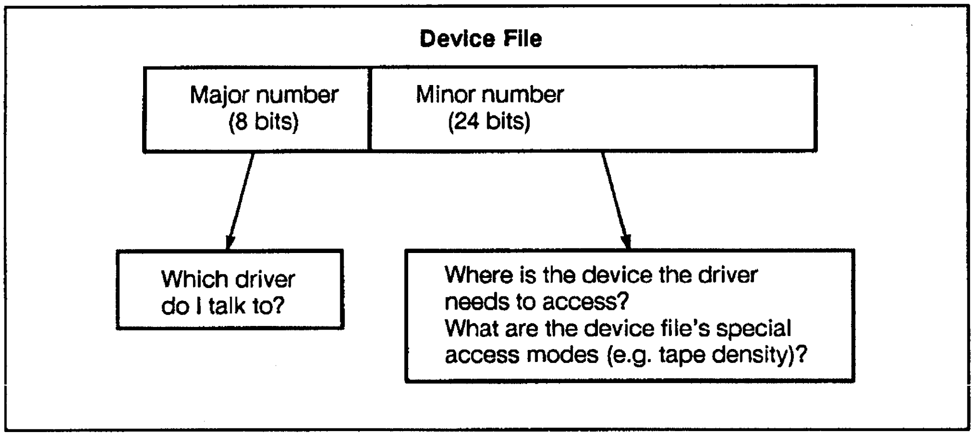


Figure 11-5. Device Files Contain Major and Minor Numbers.

Each device driver in the system is assigned a **major number**, which the kernel uses to locate the driver routine to service an I/O request.

The driver uses the **minor number** to get to the specific device and for information regarding how to handle data.

Major Numbers

The **major number** is an index for the device driver into one of two kernel tables—`bdevsw`, the block device switch table and `cdevsw`, the character device switch table. These device switch tables list driver entry points.

Drivers that support both block and character I/O (such as `scsi` disk driver and optical autochanger) have two major numbers—a block major number and a character major number—each of which can be referenced in their respective switch tables, depending on how the device is used. Or, explained from another perspective, the character device switch table finds all the functions that can be called by a character device driver (such as the `scsi` driver) in character mode.

Major numbers are listed in the `/etc/master` file, discussed earlier in this chapter.

Major Numbers and lsdev

You can execute the *lsdev*(1m) command to get the major number of the devices configured on your system.

The `-1` designation in either character or block major number field can have several meanings:

- The device might not require that kind of major number; some devices are accessable only in block or character mode.
- The driver is not configured into the kernel.
- The user may not access the driver directly. For example, the 98625 driver handles access to the high-speed HP-IB I/O card. The user may not access it directly, but must configure the cs80 driver, which can access 98625.

Here is sample output from the `lsdev` command issued on a Series 300, 400, or 700:

```
Character     Block      Driver
    0          -1        /dev/console
    1          -1        HP98628, HP98626, and HP98642
    2          -1        /dev/tty
    3          -1        /dev/mem, /dev/kmem, and /dev/null
    4           0        CS80 disk
    5          -1        HP7970/HP7971 nine-track magnetic tape
    6           1        HP9826/HP9836 internal flex disk
    7          -1        HP-UX printer
    8          -1        /dev/swap
    9          -1        HP7974/HP7978 nine-track magnetic tape
   11           2        Amigo disk
   13          -1        SRM option
   16          -1        Master pty
   17          -1        Slave pty
   18          -1        IEEE 802 device
   19          -1        Ethernet device
   21          -1        HP-IB DIL
   22          -1        GPIO DIL
   23          -1        raw 8042 HIL
   24          -1        HIL
   25          -1        HIL cooked keyboards
   26          -1        ciper printer
   47           7        SCSI disk
   54          -1        SCSI tape
   55          10        optical autochanger
```

Here is sample output from `lsdev` on a Series 800:

Character	Block	Driver	Class
-1	-1	processor	processor
-1	-1	memory	memory
-1	-1	bus_converter	bc
51	-1	lan1	lan
-1	-1	cio_ca0	cio
7	8	disc1	disk
12	10	disc2	disk
13	7	disc3	disk (SCSI)
-1	-1	hpib0	hpib
12	10	hpfl0	hpfl
50	-1	lan0	lan
1	-1	mux0	tty
5	5	tape1	tape_drive
57	11	rdu	pseudo
34	-1	ktestio	pseudo
29	-1	ktest	pseudo
60	-1	nm	pseudo
56	-1	ni	pseudo
46	-1	netdiag1	pseudo
48	-1	mirconfig	pseudo
31	-1	sy	pseudo
-1	13	sw1	pseudo
8	3	sw	pseudo
17	-1	pty1	pseudo
16	-1	pty0	pseudo
3	-1	mm	pseudo
41	-1	meas_drivr	pseudo
27	-1	dmem	pseudo
28	-1	diag0	pseudo
69	-1	devconfig	pseudo
0	-1	cn	pseudo

For more information, consult the *lsdev*(1m) and *lssf*(1m) manual pages in the *HP-UX Reference*.

Minor Numbers

The **minor number** is a six-digit hexadecimal encryption of address and characteristic hardware information, stored in the inode of the device's special file. The minor number gives the device driver the channel or connection through which to perform I/O.

Minor numbers are used to distinguish among devices whose major numbers are identical (that is, use the same driver) and are connected to the same system processor unit (SPU). The minor number typically defines one or both of the following:

- The device's physical location (hardware address or logical unit number) and switch settings, to enable the system to locate the peripheral.
- Behavioral information. The minor number's content depends on the type of device; for example, the minor number of a magnetic tape drive encodes tape density and other behavioral information.

Minor numbers also provide some flexibility in your system configuration. On Series 300, 400, and 700 systems, the minor number can be decoded by select code or function number to a specific function on a multi-function card.

Device drivers can use minor numbers to select a communication line (such as a specific terminal on a multiplexer) or operating mode (such as magnetic tape drive with different minor numbers to enable different formats). In the following long listing of two device files of a medium-density tape drive file on a Series 800,

```
crw-rw-rw-  1 bin    bin     5 0x020000 Oct  1 11:04 /dev/rmt/0m
crw-rw-rw-  1 bin    bin     5 0x0a0000 Nov  8  1989 /dev/rmt/0mn
```

`0x020000` and `0x0a0000` are both minor numbers that can be used to access the tape drive, but as you can see from the output of `/etc/lssf /dev/rmt/0m*`,

```
tape1 lu 0 bpi 1600 at&t address 4.2.3 /dev/rmt/0m
tape1 lu 0 bpi 1600 no_rewind at&t address 4.2.3 /dev/rmt/0mn
```

`/dev/rmt/0m` rewinds the tape when the programs are finished with the device, whereas `/dev/rmt/0mn` does not. Thus, if you specify `/dev/rmt/0m` in the *fbackup*(1m) command, the system will rewind the tape at the end of the backup; if you specify `/dev/rmt/0mn`, it won't.

Since minor numbers give device-driver specific information, they differ by system architecture.

Minor Numbers on Series 300/400

For the Series 300/400 systems, the digits after `0x` identify the select code and other device-specific information. (See discussion of select codes earlier in this chapter.)

The select code is preset for all device interfaces on your system. Each device interface must have a unique select code. You might need to change the select code of a device interface if the preset value is the same as another on the system. Refer to *Installing Peripherals, Volume One*, chapter 2, "Installing Interface Accessary Cards," for instructions to change the select code for a particular interface.

Other fields in the minor number are device-driver dependent. After the select code is information identifying bus or port address. The *Installing Peripherals* manual gives specific instructions for setting up devices and interface cards to a specific address, using SAM (recommended method) or manually (using `/etc/mknod`).

Table 11-2 summarizes minor number formats for Series 300/400 HP-UX peripheral devices. In all cases, `0x` indicates hexadecimal format. Unless otherwise noted, `Sc` indicates the select code of the interface, an 8-bit value.

Table 11-2. Summary of Series 300/400 Minor Number Formats

Peripheral Type	Format	Syntax
Disk drive	`0xScBaUV`	`Ba` = HP-IB or SCSI bus address (4-bit value) `U` = unit number. `V` = volume number (0 for single file systems).
Magneto-optical disk drive library	`0xScB0Sur`	`B` = SCSI bus address (4-bit value). `0` = Reserved (always 0) `Sur` = surface (16-bit value) (See *autochanger*(7) for surface values.)
Nine-track magnetic tape\ DDS-Format tape drive	`0xScBaUV`	`Ba` = bus address (from switch settings on the tape drive). `U` = unit number (0=SCSI drive, 4=HP-IB drive; 4-bits). `V` = volume number (4-bit value), as follows: bit 3—always 0 bit 2—immediate report (0=on,1=off) bit 1—close style (0=AT&T,1=Berkeley) bit 0—rewind on close (0=yes,1=no)
Cartridge tape drive	`0xScBaUV`	`Ba` = HP-IB bus address (from tape-drive switch settings). `U` = unit number. `V` = volume number (0 for single file systems).
Terminal and modem	`0xScPa0X`	`Pa` = Port address (switch settings on the device) `0` = 4 bits of 0. `X` = hexadecimal 4-bit binary number, defined as follows: bit 3—always 0 bit 2—0=modem; 1=direct connect bit 1—protocol (0=Simple/USA; 1=CCITT/Europe) bit 0—modem (0=dial-in; 1=dialout)
Pseudo-terminal device file	`0x00YYYY`	`YYYY` = unique hexadecimal value (0 to **`npty -1`**), to identify relationship between master and slave.
HP-IB plotter and digitizer	`0xScBa00`	`Ba` = HP-IB bus address (4-bit value). `00` = always 0

Table 11-2.
Summary of Series 300/400 Minor Number Formats (continued)

Peripheral Type	Format	Syntax
RS-232-C plotter	0xScPo0A	Po = Port number, values 0-3 (8-bit value). 0 = always 0 (4-bit value) A = access type, defined as follows: bit 3—always 0 bit 2—0=modem; 1=direct connect bit 1—protocol (0=Simple/USA; 1=CCITT/Europe) bit 0—modem (0=dial-in; 1=dialout)
HP-IB printer	0xScBa0M	Ba = HP-IB bus address (4 bit value). 0= always 0 (4 bit value). M = mode of operation, defined as follows: bit 3 (Auto FF)—1=NO-EJECT bit 2 (Case Fold)—1=Upper bit 1 (Overprint)—1=NOCR bit 0 (Protocol)—0=block (Amigo); 1=character
RS-232-C printer	0xScPo0A	Po = Port number, values 0-3 (8-bit value). 0 = always 0 (4-bit value) A = access type, defined as follows: bit 3—always 0 bit 2—0=modem; 1=direct connect bit 1—protocol (0=Simple/USA; 1=CCITT/Europe) bit 0—modem (0=dial-in; 1=dialout)
Parallel printer	0xScPo000	Po = Port number, values 0-3 (8-bit value). 000 = always 0 (12-bit value)
Graphic device	0xSTXXXX	S = Select code (4-bit value). T = Device resolution and physical address (12-bit value). XXXX = Device-driver information
HP-HIL device	0x0000B0	0000 = always 0 (16-bit value). B = Position on HP-HIL bus (4-bit value). 0 = always 0 (4-bit value).
GPIO device	0xSc0000	Sc = Select code (8-bit value). 0000 = always 0 (16-bit value).

Minor Numbers on the Series 700

The minor number, used to describe the hardware path to the device, is a 24-bit value with the following format:

4 bits	4 bits	4 bits	12 bits
SMB #	Slot #	Function #	Driver Specific Information

Figure 11-6. Series 700 Minor Number Format

where:

SMB # — is a four-bit single hexadecimal digit representing the System Bus Module number with the following decimal values:

0	Standard Graphics Connect (SGC) slot 1
1	SGC slot 2
2	Core I/O
4	EISA adapter
8	Processor
9	Memory

Slot # — is a four-bit single hexadecimal digit representing the EISA interface card slot number.

Function # — is the four-bit value specifying a particular function if the interface card is a multifunctional card. The basic function numbers of the Core I/O are listed:

1	SCSI
2	LAN
3	HIL
4	Serial port A
5	Serial port B
6	Parallel

If the interface is not a multifunction card, the function number contains driver-specific information. (Additional function numbers may be model-specific.)

Driver Info — 12-bit (three-hexadecimal digit) value that can be found in the *HP-UX Reference*, section 7 for the particular driver you are using.

Decoding a Minor Number for a Series 700 SCSI Device.

Table 11-3 may be useful for interpreting the information that follows.

Table 11-3. Decimal, Binary, and Hexadecimal Equivalents

Decimal	Binary	Hexadecimal	Decimal	Binary	Hexadecimal
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

The hexadecimal notation for the Series 700 minor number follows the format `0xSSFDDD`, which encodes SMB, slot number, function, and device-specific information. The minor number for a device connected to the Core I/O SCSI interface with a driver specific value of 0x333 is:

```
0x201333
```

The corresponding binary minor number value is:

| 0010 | 0000 | 0001 | 0011 0011 0011 |

These hexadecimal values can be interpreted as follows:

0010	The core I/O card occupies SMB slot is 2.
0000	The core I/O card has no slots.
0001	The SCSI function is number 1 on the core I/O card.
0011 0011 0011	The driver-specific information is 0x333.

information is 0x333.

Decoding a Minor Number for a Series 700 EISA SCSI Device.
The minor number for a device connected to an EISA SCSI interface in slot one with a driver specific value of 0x333 is:

```
0x410333
```

The corresponding binary minor-number value is:

| 0100 | 0001 | 0000 | 0011 0011 0011 |

where the SMB # is 4, the slot number is 1, and the driver-specific information is 0x333.

Note Function numbers only apply to the Core I/O interfaces and slot numbers only apply to the EISA interfaces.

Minor numbers might vary from the formula shown, particularly in the case of `pty` drivers, which have unique requirements.

Table 11-4 summarizes minor number formats for Series 700 HP-UX peripheral devices. In all cases, 0x indicates hexadecimal format.

Table 11-4. Summary of Series 700 Minor Number Formats

Peripheral Type	Format	Syntax
SCSI disk drives	0xBMFT00	BmF—Bus module and function number of controller, as follows: 201 = SCSI on core I/O 4S0 = SCSI on EISA (S = slot number on EISA bus) T—Target number (0-6 = bus address setting on device) 00—Place holders for disks
DDS-format tape drive	0xBMFTDO	BmF—Bus module and function number of controller, as follows: 201 = SCSI on core I/O 4S0 = SCSI on EISA (S = slot number on EISA bus) T—Target number (0-6 = bus address setting on device) D—Device compression, partition selection bits O—Operation bits (e.g. immediate reporting, rewind) (See *Installing Peripherals* for DDS tape D and O bit definitions.)
Magneto-optical autochanger disk-drive library	0xBMFTSr	BmF—Bus module and function number of controller, as follows: 201 = SCSI on core I/O 4S0 = SCSI on EISA (S = slot number on EISA bus) T—Target number (0-6 = bus address setting on device) Sr—Surface for media (See *Installing Peripherals* and *autochanger*(7) for defining media surfaces.)
Parallel printer	0xBMFT00A	BmF—Bus module and function number of controller, as follows: 201 = SCSI on core I/O 4S0 = SCSI on EISA (S = slot number on EISA bus) T—Target number (0-6 = bus address setting on device) 00—always 00 (12-bit value) (See *Installing Peripherals* and *cent*(7) for handshake modes.)

Minor Numbers on Series 800

The assignment of minor numbers on Series 800 is virtually foolproof: if you use `insf -d driver_name`, `insf` installs device files in the current directory with properly encoded minor numbers and logical unit numbers. `insf` is also run automatically at boot-up. The *Installing Peripherals* manual has device-specific information on minor numbers throughout its text and Appendix A provides minor number formats for all Series 800 device drivers.

The minor number on Series 800 is a six-digit hexadecimal field containing such information as:

- logical unit number (all physical devices)
- section number (disks)
- port number (terminals)
- SCSI target number (disk, tape)
- HP-IB address (plotter, disk, tape)
- device options (most devices)

As with the Series 300/400/700, Series 800 minor numbers are constructed differently for each kind of device. For example,

- The logical unit of each component daisy-chained to an HP-IB, HP-FL, MUX, SCSI, or HP-HIL interface is encoded into the peripheral's minor number.
- The minor number for a tape device specifies the density at which the tape drive is to operate and whether it should rewind at the end of a file.

Refer to the *Installing Peripherals*, device documentation, and *HP-UX Reference Manual* for information on the specific device in question.

Logical Unit Number.
The `lu` is the **logical unit** specification that allows one device driver to handle many devices. (The `lu` is necessary because the complexity of the Series 800 bus structure makes it impossible to store all hardware addressing information in the minor number.)

The `lu` is assigned by *insf*(1M). The `lu` is mapped into the I/O tree, which the kernel then consults to determine which specific hardware address the driver wants to access.

The `lu` allows the system to keep track of all configured devices belonging to the same class of device.

For example, consider a system configured with HP-IB, HP-FL, and SCSI disk drives, which are run by device drivers `disc1`, `disc2` (`disc4` for HP-PB), and `disc3` respectively. If a system has 10 disk drives, each one is assigned an `lu` number from 0 to 9. All three kinds of disk drive (HP-IB, HP-FL, and SCSI) share the same pool of `lu` numbers, because all are of the same class of devices (`disk`).

`lu` numbers are assigned in chronological order, according to *when* the device was added to the system rather than by architectural proximity, although the system's automatic configuration capabilities associate the device `lu` with a low-to-high ordering of slot numbers in hardware.

The `lu` is associated with hardware paths in the I/O tree and in `/etc/ioconfig`. `/etc/ioconfig` is used to maintain logical unit number to hardware path mappings across system boots.

The *Installing Peripherals* (Series 800) manual contains information about logical unit numbers.

Sources for Minor Number and other Device Information

Other bits in Series 800 minor numbers provide behavioral information or location about the device, including section number, MUX port, device options, and other flags. Since each is directly related to the kind of device accessed, consult the device's manual page in the *HP-UX Reference* for specific details. Sometimes you might need to make device files for individual devices or classes of devices.

The following manual pages in the *HP-UX Reference* give minor-number and device-file conventions for specific classes of devices:

- *intro*(7)—nine-track tape and disk devices
- *autochanger*(7)—SCSI optical mass-storage devices (Series 300/400/700 only)
- *cent*(7)—parallel printers (Series 300/400/700 only)
- *ct*(7)—cartridge-tape devices
- *disk*(7)—disk devices
- *floppy*(7)—direct flexible disk access (Series 700 only)
- *graphics*(7)—CRT graphics devices (Series 300/400/700 only)
- *hil*(7)—HP-HIL device driver
- *hpib*(7)—HP Interface Bus driver (Series 800 only)
- *iomap*(7)—physical address mapping (Series 300/400 only)
- *lp*(7)—line printer (Series 300/400/800 only)
- *mt*(7)—magnetic tape, DDS-cartridge tape, and QIC-cartridge tape interface and controls
- *scsi*(7)—Small Computer System Interface device drivers
- *scsi_changer*(7)—SCSI media changer device driver (Series 700 only)
- *scsi_ctl*(7)—SCSI device control disk driver (Series 700 only)
- *scsi_disk*(7)—SCSI direct access disk device driver (Series 700 only)
- *scsi_tape*(7)—SCSI sequential access tape device driver (Series 700 only)

Device File Name Conventions

The *intro*(7) and manual pages of the *HP-UX Reference* describe naming conventions for various types of device files, as they are shipped on your system. Using the naming conventions makes your system easier to support and maintain. Also, some commands (such as the *insf*(1m) command on the Series 800 and SAM) expect the recommended path names and file names.

Critical elements of a device file are its major and minor numbers. Particularly on a small system, you can name device files as you wish. If you do so, keep the names of your `/dev` device files intuitive. For example, printer device files are easily identified when named by their product, such as `/dev/ptr2567C` for an HP 2567C printer. Similarly, the subdirectory name `/dev/ac` or `/dev/autochg` would represent autochanger.

Naming Disk Devices

Device files for disk drives should reside in the `/dev` directory. Disk drives require both character and block device files. Disks use the `/dev/rdsk` directory for the character device files and the `/dev/dsk` directory for the block device files.

An exception exists for magneto-optical devices, which use the `/dev/rac` and `/dev/ac` directories. Description of file-name conventions for magneto-optical devices follows standard disk devices in this section.

File names for disk devices on Series 300/400 use the following format:

```
/dev/[r]dsk/c#d#[l#]s#
```

where:

`r`	Indicates a raw (character) interface to the disk.
`c#`	Specifies the controller number. The `#` should be replaced with a capitalized hexadecimal representation of the select code.
`d#`	Specifies the bus (target) address, a number representing the entire device.
`[l#]`	The logical unit number (`l`) is used to identify specific units in integrated devices.

`s#` The `s#` stands for section number. This is typically zero, except when using software disk striping (refer to the *System Administration Tasks* manual for information).

File names for disk devices on Series 700 use the following format:

`/dev/[r]dsk/c###d#[l#]s#`

where:

`r` Indicates a raw (character) interface to the disk.

`c###` Specifies the controller bus module and function number; the `###` can be:
- 201 = core I/O (the default)
- 4# = EISA bus (4) and slot number (#). If specifying a device on an EISA bus, only two (not three) digits follow the `c` (for example, `41`).

`d#` Specifies the target number, the address set on the device itself. This number represents the entire device.

`[l#]` Disk arrays require a logical unit number, which designates specific device units.

`s#` The `s#` stands for section number. This is typically zero, except when using software disk striping (refer to the *System Administration Tasks* manual for information).

On Series 800 systems, default disk device files are shipped as follows:

`/dev/[r]dsk/c#d#s#`

where:

`r` Raw interface to the disk.

`c#` Controller, represented by the `lu` number.

`d#` Drive number; always 0.

`s#` Section number.

The assignment of controller, drive, and section numbers is described in the *System Administration Tasks* manual. Additional information about disks can be found in `disk(7)` in the *HP-UX Reference Manual*.

Naming Magneto-Optical Disk Devices

Magneto-optical disk devices use the /dev/rac and /dev/ac directories for character and block device files, respectively. Device files for magneto-optical disk devices on HP 9000 systems use the following format:

/dev/[r]ac/c#d#_pp

where:

r	Indicates a raw (character) interface to the disk.
c#	Specifies the controller bus module and function number; the # can be: Capitalized hexadecimal representation of select code (Series 300/400) 201 = core I/O (Series 700 default) 4# = EISA bus (4) and slot number (#). Logical unit number (Series 800)
d#	Specifies the target number, the address set on the device itself. This number represents the entire device.
_	Underscore (always present).
pp	Disk platter number and side (a or b).

For further information, consult the chapter, "Setting up Devices Using HP-UX Commands," in the *Installing Peripherals* for your system. Also see *autochanger*(7) in the *HP-UX Reference Manual*.

Naming Nine-Track Magnetic Tape Drives

The following naming convention is recommended for magnetic tape devices because it connects most of the mode flags with the device name:

/dev/[r]mt/c#d#[hml][c][n]

where:

r	indicates a raw (character) device.
c#	indicates the controller number (optionally specified by the system administrator).
d#	is the device number.
hml	indicates the density:

- h (high) for 6250 bpi.
- m (medium) for 1600 bpi.
- l (low) for 800 bpi.

`c` indicates data compression.
`n` indicates no rewind on close.

For example, `/dev/rmt/2mn` is raw device 2 at 1600 bpi with no rewind and no compression.

Additional information about 9-track magnetic tape is available in *mt*(7) of the *HP-UX Reference Manual*.

Naming DDS-Format Tape Drives

Device files for DDS-format tape drives are named as follows:

```
/dev/rmt/c#d#m|c[n]
```

where:

`r`	indicates a raw (character) device.
`c#`	specifies the controller number, in capitalized hexadecimal notation.
`d#`	is the device number.
`m|c`	indicates density. Use either `m` for medium density (standard DDS format) or `c` for compressed density.
`n`	indicates no rewind on close.

Additional information about DDS cartridge tape can be found in *mt*(7) of the *HP-UX Reference Manual*.

Naming QIC-Format Tape Drives

QIC format is an industry standard for the quarter-inch cartridge (QIC) tape drives, available as SCSI devices on Series 700/800 systems. QIC-format device files are named as follows:

```
/dev/rmt/#m[n]
```

identical to DDS format, or

```
/dev/rmt/#qic
```

for default AT&T-style QIC format, or

```
/dev/rmt/#qic[120|150|525][n][b]
```

r	indicates a raw (character) device.
#	is the SCSI logical unit number.
120|150|525	indicates specific QIC format, if needed.
n	indicates no rewind on close.
b	indicates Berkeley style (if b is omitted, indicates AT&T style)

For further information about QIC-format cartridge tape files, see *mt*(7) or *insf*(1M) of the *HP-UX Reference Manual*. For information on backing up your system to QIC cartridge tape, consult "Backing Up and Restoring Your Data" of *System Administration Tasks* manual for your system.

Naming Cartridge Tape Drives

As shipped, device files for cartridge tape drives are named as follows:

```
/dev/[r]ct/[r]c#[d#][s#]
```

where:

r	indicates a raw interface to the cartridge tape; the second r is reserved to indicate that this cartridge tape is on a remote system.
c#	indicates the controller number.
d#	optionally indicates the drive.
s#	optionally indicates a section number.

The assignment of controller, drive, and section numbers is described in the *System Administration Tasks* manual; also see in *ct*(7) of the *HP-UX Reference Manual*.

Naming HP-IB Devices (Series 800 only)

As shipped, HP-IB device files are named as follows:

```
/dev/hpib/#[a#]
```

where:

`#`	the first `#` specifies the bus number (assigned by the system administrator)
`a#`	if present, `a#` specifies the address on the bus. Device files without an address suffix denote the raw bus. See *hpib*(7) for full explanation of auto-addressed files vs. raw bus files.

Naming Modem Device Files

Modem device files are tricky because you must create several of them, depending on which software release you run. If running the current software release, you must create two modem device files:

`/dev/ttydNN`	dial-in terminal
`/dev/culNN`	dial-out line
`/dev/cuaNN`	dial-out line

In HoneyDanBer `UUCP` and `cu`, the system uses `/usr/lib/uucp/Systems` to dial the connection.

If you are using earlier software release or a non-HoneyDanBer `uucp`, the system dials its connection using `/usr/lib/dialit.c`, which requires you to create a third modem device file:

`/dev/cuaNN` auto-dial unit

Besides the peculiarities of modem device file names, you might pay special attention to the modem minor numbers. The last two digits of the minor number indicate whether a terminal is hard-wired or connected by modem. Normally, you have a modem on one port and your terminals on different ports (for example, a modem on port 0 and a dial-in terminal on port 1).

Some devices have very specific minor-number requirements. For example, the following table shows the minor number for a modem connection at select code 9:

Type	Device file name	Minor number
dial-in	`/dev/tty01`	`0x090000`
dial-out	`/dev/cul01`	`0x090001`
dial-out	`/dev/cua01`	`0x090001`

Note that `/dev/cua`*NN* is used to establish the connection, whereas `/dev/cul`*NN* is used once the connection is established. You can link them together, as follows:

```
# ln /dev/cul01 /dev/cua01
```

Direct-connect devices also have special needs. The minor number must end in the digit `4`. If you have a MUX card, the minor number for a direct-connect at port 1 would be `0x0d0104`:

Type	**Device file name**	**Minor number**
direct-connect	`/dev/ttyd01`	`0x0d0104`

12

HP-UX Peripherals

Computers are used to process massive amounts of data, which must be stored as large files online. A variety of peripheral devices meet this storage need. This chapter describes the following technologies that handle and store magnetic media:

- Nine-track magnetic tape drives.
- DDS-format tape drives.
- Cartridge tape drives.

This chapter also describes:

- The CD-ROM file system.
- Rewritable optical disk drives.
- Terminals and modems.
- HP-IB and SCSI guidelines.

Magnetic Tape

Note Series 700 systems do not support nine-track magnetic tape drives; see discussion of DDS-format tape drives instead.

Perhaps the closest thing to an industry standard for mass media, nine-track (1/2-inch) magnetic tape ("magtape") serves as a low-cost, high-capacity medium to store information. Magnetic tape is also the most interchangeable medium between different hardware and operating systems.

Magnetic Tape Physical Characteristics

Magnetic tape is a medium similar to everyday audio cassette tape, but it is used to store digital information. All standard magnetic tape is 1/2-inch wide, and is available in 600, 1200 and 2400 foot reels. (A 2400 foot reel is approximately 1 foot in diameter.) The size of the reels, hubs, tape width and other mechanical properties are all specified by ANSI standard.

On the back of the reel is a removable soft plastic **write ring.** Every magtape drive has a sensor mechanism to detect the presence of this ring. When a ring is present the tape can be written to by the host and cannot be written when absent (it is **write protected**). Normally, once a tape is written, the ring is removed and left out indefinitely except when being rewritten.

Tape Density

The measure of the amount of information which can be stored in a given area of tape is known as **tape density. Bits per inch** (bpi), a common measure of tape density, is the number of bits per track, recorded per inch on the tape. For nine-track tape, eight data bits and one parity bit are written across the width of the tape simultaneously. Thus for nine-track tape, bpi is synonymous with **characters per inch** (cpi). One of these characters is sometimes called a **frame.**

Logical Organization of Nine-Track Magnetic Tape

The **load point**, or beginning of tape, is a foil mark placed about 10 feet from the beginning of a tape. The foil mark is a short piece of silver tape placed on one edge of the tape on the non-recorded side by the manufacturer.

When you load a tape (put the tape in the drive, and press "load"), the drive searches forward until the load point is found and placed under the sensor. The first write is then treated specially: several inches of tape are skipped and then, when using PE or GCR formats, a special burst of data is written to the tape (which is invisible to the user). This is the **identify burst**. Data is recorded after the identify burst in the usual way. The first read expects the identify burst, and quietly skips over it. Some smart drives, such as the HP 7978, can determine the tape density from the identify burst (1600 and up).

File Marks

A special command to the drive writes a **file mark** to the tape. A file mark is a special type of record recognized by the drive and reported as a Boolean condition during reading. It is not possible to write a file mark as ordinary data.

Single file marks are used to separate logical files on tape. Two consecutive file marks are used to signify the logical EOT. Data is undefined past the logical EOT.

End of Tape Markers

There is both a logical end of tape (EOT) and a physical EOT (see Figure 12-1). Logical EOT is two consecutive file marks. Physical EOT is a foil mark about 25 feet from the end of the reel. Like the load-point foil mark, the physical EOT foil mark is placed on the tape by the manufacturer.

Note that the distance between the EOT detector and the read/write head may vary among different model tape drives. So, one drive may return an EOT indication associated with the 1000th record on the tape, while another drive may return an EOT indication with the 999th or the 1001st record. For small records this variation may be large; for large records this variation is probably small.

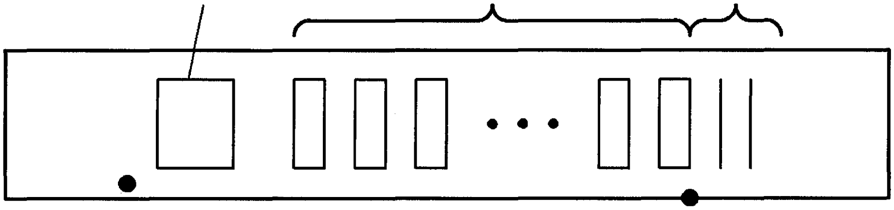


Figure 12-1. Magnetic Tape Format

Magnetic Tape Capabilities

A tape drive can read and write to the media, rewind to the load point, forward or back space one record, and forward or back space to the next file mark. Tape drives also rewind to **unload**: the tape is rewound and taken off line. Some tape drives actually rewind the tape out of the threading path; others simply set an interlock that requires manual intervention to release the tape.

When digital information is written to a tape, it is written in a series of tracks (like an eight-track car stereo). Most magtape today is written in a nine-track format. Older systems wrote six tracks plus a parity bit, resulting in 7 tracks.

Records and Record Sizes

A series of frames written to the media is known as a **record**. The physical record size is variable. The maximum limits on record size range from 16 KB to 60 KB, depending upon the tape drive. Beyond these limit, the drive rejects the request and there are *no* write/read retries. The maximum record sizes are:

HP 7971	1600 bpi—32 KB
HP 7974	1600 bpi—16 KB 800 bpi—16 KB
HP 7978A	1600 bpi—16 KB 6250 bpi—64 KB
HP 7978B	1600 bpi—32 KB 6250 bpi—60 KB
HP 7980	1600 bpi—64 KB 6250 bpi—64 KB

12

Cyclic Redundancy Check

When writing a tape, a number of frames are written by the drive in a single transaction. This collection of frames is called a **record**. Part of the record, but invisible to the user, is a **cyclic redundancy check** (CRC). The CRC is recorded as some additional frames on the tape. There is a very short blank section between the true record and the CRC. Following the CRC is a nominal 1/2-inch gap of unrecorded tape, known as the **inter-record gap** or IRG. The next record follows the gap. If either the frame parity or the CRC is incorrect when the tape drive reads the tape, an error is generated by the drive. Newer formats (1600 bpi and above) generate a preamble and postamble to help synchronize the read logic.

Coding

Tape is recorded in several ways. Older systems use **Non Return to Zero Immediate** (NRZI) coding, and record with a tape density of either 200, 556, or 800 bpi (bits per inch). Newer tapes use **Phase Encoding** (PE) and record at 1600 bpi, or they use **Group Coded Recording** (GCR) and record at 6250 bpi. There may be other forms of coding as well, but these are the most common. The HP 7971 supports a density of 1600 bpi, the HP 7974 and HP 7979 support both 1600 bpi and (optionally) 800 bpi, and the HP 7978 and HP 7980 magnetic tape drives support a density of 1600 and 6250 bpi.

The higher the density, the more information can be stored on a tape. On a 2400 foot tape, an HP 7974 at 800 bpi can only store 22 MB of data, at 1600 bpi the HP 7974 can store 43 MB, while an HP 7978 storing at 6250 bpi can write up to 140 MB of data to a tape at a rate of up to 16 MB per minute.

Write/Read Errors

Tape, in its usage for long-term archive and data interchange, is more error-prone than disks. When your tape drive is reading from or writing to a tape, and it detects an error, the normal procedure is to backspace the tape over the record and retry the tape operation. An error message is reported to the user only after the driver gives up. Many more tape errors are caused by dirty tape heads than by real recording errors, so you should periodically clean your tape drive as outlined in its service manual.

Tape drives do a form of reading-while-writing, and if the data is not properly recorded, an error will be detected. The normal procedure is to backspace and retry writing the record once, and if that fails, to backspace, write a long gap and try again on a section of tape farther down. A **long gap** is several inches of erased tape. That's why we said an IRG is "nominally" 1/2 inch long.

Preventive Maintenance

There are several maintenance procedures for tape. A tape can be completely erased (degaussed), or the beginning of the tape can be discarded and a new load point put on (stripped). There is also a tape cleaning and certifying machine that will knock off any loose oxide and check that the tape will record properly over its full length (certified). This always makes any data on the tape unusable. Commercial shops certify their tapes fairly often, and discard them if they get too short or fail to certify. It is also an excellent idea to clean the tape head and guides of your drive periodically as they tend to accumulate loose oxide and other contaminants.

Tape Streaming

Tape drives are designed to transfer data by one of two methods—**start/stop** or **streaming**.

The HP 7971 is a start/stop tape drive, and is designed to stop and restart the tape fairly quickly. The HP 7971 transfers data with very little buffering between the computer and the tape drive's read/write head; the drive must stop the tape between records, and wait for the next record.

The HP 7974, HP 7978, and HP 7980 are streaming magnetic tape drives.

A streaming magnetic tape drive is designed to move continuously, reading data from a buffer or writing data to a buffer, not stopping between records, as start/stop tape drives do. Streaming increases the rate at which a tape drive can write data onto tape. Before a tape drive can write data onto a tape, the drive read/write head must be positioned at the proper place on the tape, and the tape must move across the head at the proper speed. After writing a record on the tape, if a streaming drive has already received the data for the next record from the computer, it can continue to move the tape across the head without slowing down to write the next record.

If the drive has not received the data for the next record after writing a record on the tape, then the drive must reposition the tape. This involves stopping the forward motion of the tape, backspacing the tape to some point preceding the beginning of the next record to be written, stopping the tape, and waiting for your computer to send the data for the next record. *The average data transfer rate is much higher when the drive streams than when it repositions*, especially for the HP 7978. The HP 7974 supports both a start-stop and a streaming mode. The HP 7978 supports only a streaming mode. Both drives are much faster than the HP 7971 when they stream. When they do not receive data fast enough to stream, the HP 7974's performance is similar to the HP 7971; the HP 7978 and HP 7980 are much slower.

Immediate Response Mode

To help your computer send data fast enough to permit the drive to stream, the HP 7974, HP 7978, and HP 7980 support **immediate response mode**. Ordinarily the actions of your computer and the drive are serialized. Your computer sends data to the drive. Then the drive writes the data to the tape. After the data is written, the drive returns status information to the host indicating whether the write succeeded or failed. When immediate response is enabled the drive returns status before it writes the data to the tape.

This is accomplished by the drive buffering the data it receives from your computer in high speed memory which is built into the drive. The transfer rate between the host and this buffer memory is much faster than the transfer rate would be if the drive transferred the data directly to the tape. Because the drive returns status to your computer very quickly, the host's and the drive's activities overlap, so the average transfer rate to the drive has a much better chance of being fast enough to permit the drive to stream. Even when the drive has to go through a reposition cycle, it can still be buffering additional records from the host.

Even with immediate response enabled, the HP 7974 and HP 7978 tape drives typically don't stream continuously because the programs running on the Series 300 don't collect their data from the disk fast enough to supply it to the tape drive. However, they still perform faster than the HP 7971 stop/start tape drive.

An identical concept applied to CS/80 cartridge tape is referred to as immediate report.

DDS-Format Tape Drive

Digital data storage (DDS), based on digital audio-tape technology, provides a means for storing large amounts of data more compactly than on nine-track magnetic tape and faster than on cartridge tape. A single DDS tape cassette for a HP Series 6400 Model 1300S tape drive can hold up to 1.3 gigabytes of data.

Comparison of DDS-Format and Nine-Track Magnetic Tape

- data coding: DDS-format and nine-track magnetic tapes use different coding format.
- EOD: When a DDS-format drive writes a sequence of records to tape and then rewinds, it inserts an EOD area at the end of the recrods. Subsequent reads ignore any data after the EOD area. Nine-track tape drives do not perform any comparable activity.
- Write-protecting media: DDS-format cassettes use a small write-protect tab to protect the media from being overwritten. Nine-track tapes use write-protect rings.
- Online/Offline: Online/offline mechanisms are similar for both technologies. If a cassette is loaded in a DDS-format drive and the device is power-cycled or HP-UX is rebooted, the mechanism will cause the drive to go offline. To bring it back online, eject the cassette and reload it.
- Streaming and immediate report: DDS-format drives support both streaming and immediate report modes.

The DDS-format drive responds to all HP-UX commands appropriate to magnetic tape devices, including `cpio`, `tar`, and `mt`.

Logical Organization of DDS-Format Tape

Both DDS-format tapes and nine-track magnetic tapes can be viewed as a sequence of records and tape marks. Two concluding tape marks represent logical EOD (End of Data) for both formats.

The following example illustrates the sequence of events that occur during a transaction with the drive:

When a tape is loaded, the DDS-format drive going online and positions the tape at LBOT (Logical Beginning of Tape). In this example, two commands are issued:

12

```
tar cf /dev/rmt/0mn dir_A
tar cf /dev/rmt/0m dir_A
```

The first `tar` command causes a sequence of 10-KB records to be written. When the archive is finished, the driver automatically writes two filemarks (since the last operation is a write) and backspaces one file mark (since the selected device was `0mn`, for no rewind on close). The tape head is now positioned between the two filemarks when the second `tar` operation is called. The second sequence of tape records is written (writing over the second filemark). When the second archive is completed, the driver's close routine writes two filemarks and rewinds (`0m`) the tape to LBOT. The tape then logically looks like this:

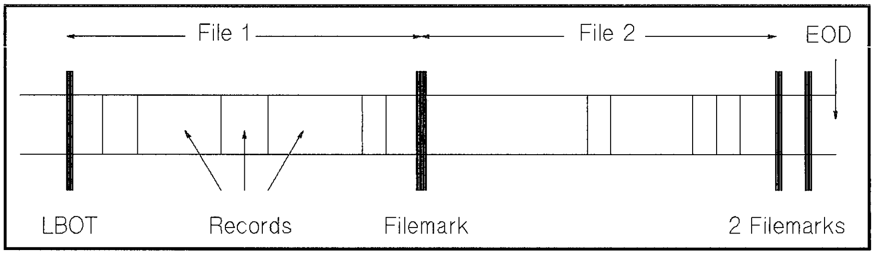


Figure 12-2. Organization of DDS-Format Tape

Tape archives such as these are sometimes called filesets, since they usually consist of many files. A tape can consist of one or more files or filesets, with the files separated by single filemarks. The last file is followed by a double filemark, representing EOD. The `mt` command is used to space between the files.

For More Information

For information on DDS-format tapes and their operation, refer to the following resources:

- User's manual for the HP Series 6400 Model 1300 DDS-Format Drive (available as 1300H for HP-IB and 1300S for SCSI).
- *Installing Peripherals* manual, for information on customizing HP-UX for a DDS-format drive.
- *mt*(7) manual page in the *HP-UX Reference*.

12

Cartridge Tape Drives

In addition to nine-track magnetic tapes and DDS-format tapes, Hewlett-Packard manufactures a series of 1/4-inch data cartridge tapes used for the installation and updates of HP-UX. The cartridge tapes can also be used for inexpensive backups. These data cartridges, model HP 88140, have most of the benefits of nine-track magnetic tape but are cheaper and easier to handle. However, they do not offer the same level of data interchange between non-HP-UX machines as the nine-track or DDS-format tapes and are much slower to use.

Like nine-track magnetic tape drives, cartridge tape drives can stream data in immediate response/report mode. See the discussion of "Immediate Response Mode" earlier in this chapter.

HP 9144/45 cartridge tape drives are supported on Series 700 systems. They require an optional EISA HP-IB card, customization of your default `dfile` with the HP-IB device drivers, and appropriate device files.

CD-ROM File System (CDFS)

CD-ROM is an acronym for Compact Disc-Read Only Memory, a technology for mass distribution and easy retrieval of large amounts of information. Compact Discs (or CDs) contain approximately 550 megabytes (MB) of data per disk. The information on the CD is virtually permanent; you can read data from a CD, but cannot write to it. Data on CDs is prepared and mastered using a specialized publishing process.

In addition to the information contained in this section, you can learn more about CD-ROM technology from these HP manuals:

- *CD-ROM: The Basics*, part number 5954-9709—discusses the introductory concepts of CD-ROM technology.
- *HP Series 6100 Model 600/S CD-ROM Drive User's Guide*, part number A1999-90002—describes the HP Series 6100 Model 600/S SCSI CD-ROM Drive.
- *HP Series 6100 Model 600/A HP-IB CD-ROM Drive User's Guide*, part number C1707-90000—describes the HP Series 6100 Model 600/A HP-IB CD-ROM Drive. (Note, the HP-IB CD-ROM drive is not supported on the Series 700.)

Overview of Implementation

This section discusses the file system format for CD-ROM data, and the standards which define it.

CD-ROM Standard Documents

Currently, there are two standard documents which define the file system format for CD-ROM. One was produced by the CD-ROM Ad Hoc Advisory Committee (popularly called the High Sierra Group, abbreviated HSG). The standard document produced by this group is called *The Working Paper for Information Processing (Volume and File Structure of Compact Read Only Optical Discs for Information Interchange)*. This document is available from the National Information Standards Organization (NISO).

The second standard evolved from the HSG standard and was produced by the International Organization for Standardization (ISO). The name of the standard document is *Information Processing (Volume and File Structure of CD-ROM for Information Interchange* (ISO 9660:1988(E)).

The file system formats described by these two standards differ in some areas, but are largely identical. HP-UX supports CD-ROM media that conform to either specification, hiding the differences from you.

System Call Changes

Because the CDFS format is so different from that of HFS, significant changes had to be made to the kernel in order to support it. These changes were made in such a way as to hide low-level details from the user. The result is that all system calls have the same calling syntax and the same behavior as they did on read-only HFS. All system calls work transparently with CDFS, exactly as they do with HFS. Those calls which cannot complete successfully on read-only media, such as *write*(2), fail gracefully; they return `errno=EROFS`.

One new system call, `fsctl`(2), was added to enable users to get CDFS-specific information from a CDFS volume. Even though `fsctl` currently supports only CDFS file systems, its architect will permit its functionality to be expanded to include HFS (and any other file systems that may eventually be supported).

The calling syntax for `fsctl` is:

`fsctl(`*fildes, command, outbuf, outlen*`)`

where *fildes* is an open file descriptor for a file on a CDFS volume, *command* is a value specifying the information desired, *outbuf* is a buffer in which the information is placed, and *outlen* is the length of *outbuf*. The available *commands* are:

`CDFS_DIR_REC` Returns the directory record for the file or directory indicated by *fildes*. Maximum size for *outbuf* is 256 bytes.

`CDFS_XAR` Returns the extended attribute record, if any, for the file or directory indicated by *fildes*. Maximum size for *outbuf* is 64 KB.

`CDFS_AFID` Returns the abstract file identifier for the primary volume whose root directory is specified by *fildes*, terminated with a NULL character. Maximum size for *outbuf* is 38 bytes.

CDFS_BFID — Returns the bibliographic file identifier for the primary volume whose root directory is specified by *fildes*, terminated with a NULL character. Maximum size for *outbuf*: 38 bytes.

CDFS_CFID — Returns the copyright file identifier for the primary volume whose root directory is specified by *fildes*, terminated with a NULL character. Maximum size for *outbuf*: 38 bytes.

CDFS_VOL_ID — Returns the volume ID for the primary volume specified by *fildes*, terminated with a NULL character. Maximum size for *outbuf*: 33 bytes.

CDFS_VOL_SET_ID — Returns the volume set ID for the primary volume specified by *fildes*, terminated with a NULL character. Maximum size for outbuf: 129 bytes.

For example, suppose the volume ID for a CDFS volume is desired. The following code retrieves this information:

```
#include <sys/types.h>
#include <sys/vfs.h>
#include <fcntl.h>
#include <sys/cdfs.h>
#include <sys/cdfsdir.h>
extern int errno;
main()
{
    int fildes, ret;
    char outbuf[129];

    fildes = open("/cdrom", O_RDONLY);

    if((ret = fsctl(fildes, CDFS_VOL_ID, outbuf, 129)) < 0) {
        printf("Fsctl failed (%d), errno = %d\n", ret, errno);
        exit(1);
    }

    ...
}
```

Note that for the following commands which retrieve volume-wide information:

`CDFS_VOL_ID, CDFS_VOL_SET_ID, CDFS_BFID, CDFS_CFID, CDFS_AFID`

fildes should be an open file descriptor for the root directory of the volume.
12 For the others (CDFS_DIR_REC, CDFS_XAR), *fildes* should be an open file descriptor for the directory or file of interest.

Command Changes

Compared to the kernel effort, only minor code changes were necessary in the HP-UX commands. This is due to the fact that most of the low-level details are hidden within the system call and kernel work, thus allowing commands to function independently of file system type.

Most HP-UX commands work transparently under CDFS. A few commands, however, are not applicable to CDFS volumes, either because of the read-only nature of the medium, or because they are inherently HFS-oriented. These commands are unsupported under CDFS, and include the following:

`clri`(1M)
`convertfs`(1M)
`diskusg`(1M)
`fsck`(1M)
`fsclean`(1M)
`fsdb`(1M)
`mediainit`(1M)
`mkfs`(1M)
`ncheck`(1M)
`newfs`(1M)
`tunefs`(1M)

No new commands have been added as a result of CDFS support.

Enabling CDFS

Before being able to mount and use CDFS volumes, the appropriate kernel code must be configured into the kernel.

Series 300/400/700 Configuration for CDFS

To add the CDFS code to the Series 300/400/700 kernel, add the `cdfs` keyword into the appropriate `dfile`. If the SCSI driver is needed for the CD-ROM drive that's been chosen, add the `scsi` keyword as well. Reconfigure the kernel using `config` and `make`, or the `sam` program. 12

A block special file is needed to mount the CDFS volume. If desired, a character special file can also be created for raw access to the CD-ROM data.

Series 800 Configuration for CDFS

To configure the Series 800 kernel, edit the appropriate `S800` file and add:

```
include cdfs;
```

Also, make sure a location is available for the HP-IB CD-ROM drive in the I/O configuration portion of the `S800` file. Regenerate the kernel using *uxgen*.

A block special file is needed in order to mount the CDFS volume. If desired, a character special file can also be created for raw access to the CD-ROM data.

HP-UX Usage

Once the kernel has been reconfigured and the necessary device files created, the CDFS volume can be mounted. Mounting a CDFS volume is done in exactly the same way as mounting an HFS volume.

Once mounted, HP-UX commands can be used to list, edit (without making changes), print, or copy files on the CDFS volume. Any executable files on the CDFS volume can be directly executed on HP-UX. Other CDFS or HFS volumes can be mounted on top of the mounted CDFS volume. In short, anything that does not require modifying the data on the CDFS volume may be done.

Many things can also be done programmatically. Files may be opened, read from, *stat*ed, checked for accessibility, etc., via the normal HP-UX system calls and library routines. Again, only those library routines and system calls which do not require modifying the CDFS volume data can be used successfully.

Optical Technology

Rewritable Optical

Traditionally, mass storage solutions have fallen into one of two categories—primary or secondary storage. Primary storage is typically one or more fixed magnetic hard disks. It is fast, random-access storage with moderately high capacity and is used as the online system disk.

Primary storage is used to store applications, heavily accessed databases, or files on which you are currently working and using extensively. It is also used for applications needing virtual memory, fast data processing, report generation, and complex calculations.

Secondary storage has consisted of one or more offline storage devices—usually a 1/4-inch or 1/2-inch tape drive, or a flexible disk drive on smaller systems. These devices are used primarily to back up the system disks and are also used for logging transactions, distributing software, archiving historical data, and exchanging data between systems.

There's a gap between primary and secondary storage in terms of access time and cost per megabyte. Average access time for hard disks is measured in tens of milliseconds. Access time for information on tapes is measured in tens of seconds for a mounted tape, minutes to hours for a tape in the library. The cost of magnetic disk storage is a few dollars, while tape storage costs a few cents per megabyte.

Rewritable optical fills this gap. With an average access time in the .1- to 10-second range, and a cost of a few cents per megabyte more than tape storage, optical drives create a new layer in the storage hierarchy called **Direct Access Secondary Storage (DASS)**. Figure 12-3 illustrates this concept.

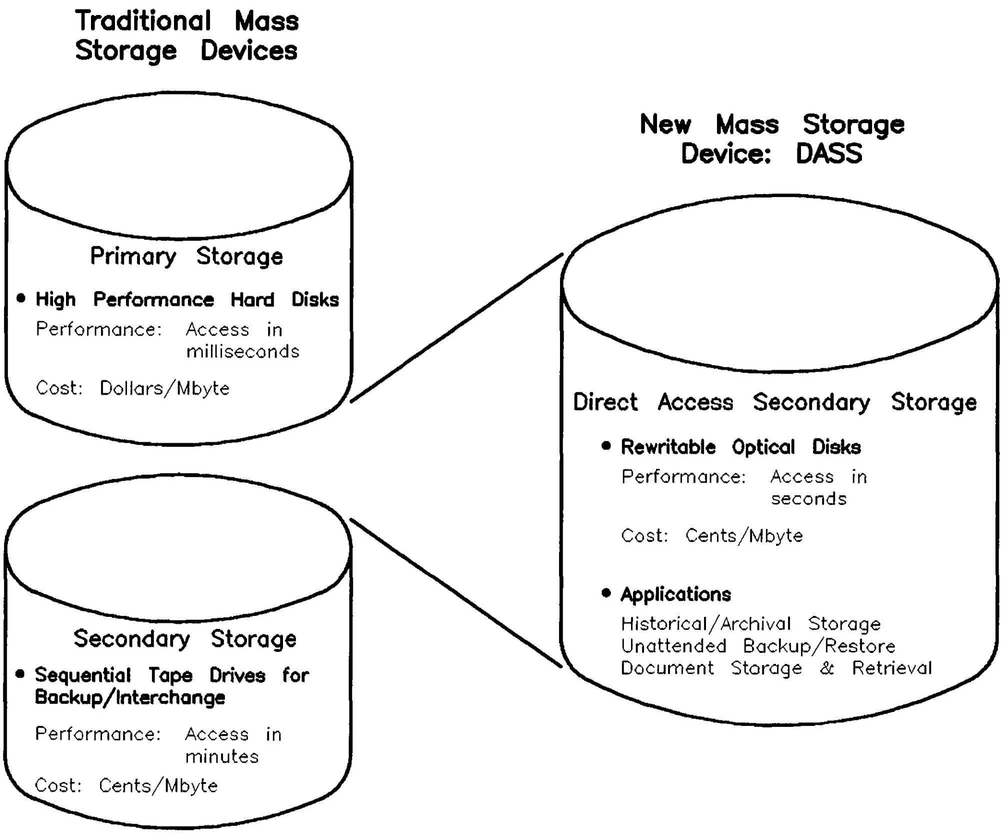


Figure 12-3. Direct Access Secondary Storage

Why Use Rewritable Optical?

Rewritable optical technology has many strengths. The disks are more durable than other media, have greater storage capacity, are more reliable, are removable, and cost far less per megabyte than magnetic disks. Rewritable optical disks are good when you need direct access to a large amount of traditionally "offline" information such as archival or backup files.

The major disadvantage of optical drives is an access time slower than that of hard disks. Due to the weight of the optical head, with its laser, lenses, and mirrors, today's access times for optical disks are 2 to 5 times slower than high performance magnetic hard disks. Therefore, the optical disk library system would not be good as a hard disk replacement.

Hewlett-Packard's Rewritable Optical Products

The Hewlett-Packard rewritable optical products range from a standalone drive for small systems to an autochanger multi-disk setup for large systems and networks. The following products are supported on HP-UX 6.5 and subsequent releases of HP-UX.

HP Series 6300 Model 650/A - Optical Disk Drive

The Model 650/A is a standalone rewritable optical disk drive. It uses 5.25 inch (130 mm) magneto-optical disks (a type of rewritable optical technology that complies with the continuous composite (C*C) format). One optical disk holds 650 MB of data (325 MB per side). However, because there is only one read/write head assembly in the drive, you must eject the cartridge and flip it over to access the second side.

Using a 2400-rpm rotational speed, the Model 650/A achieves a data transfer rate of at least 340 KB per second. It connects to the host system with a SCSI interface and can be accessed by SCSI commands as a conventional magnetic disk drive.

You can use an optical disk drive like any other disk drive. You can rewrite data an unlimited number of times and can store the disks for at least 10 years making data storage worry-free. Unlike 1/2-inch tape that you must re-tension to keep archived files readable, MO disks are maintenance-free and very durable. In addition, because of their small size MO disks require less storage space than 1/2-inch tapes.

Optical Disk Drive Guidelines

The Model 650/A can be used as a temporary or emergency boot device. If you use the optical disk drive as a boot device, you must insert the disk before you turn on the system, and you must not remove the disk until after you turn off the system.

If you use the optical disk drive as a part of your mounted file system, do not eject the disk until you have unmounted it.

HP Series 6300 Model 20GB/A - Optical Disk Library System

The Model 20GB/A Optical Disk Library System (ODLS) is a device which allows automatic, convenient access to vast amounts of information. The disk library system can have a library of up to 32 magneto-optical disks, offering a total capacity of 20.8 gigabytes.

The disk library system consists of several elements: one or more optical drives, slots to store the 32 disks, a mechanical picker, and a mail slot which lets you insert and remove disks.

Within the disk library system cabinet the mechanical picker, guided by the host computer, selects, moves, rotates, and inserts disk cartridges into the drive mechanism(s). Figure 12-4 illustrates the disk library system concept.

Since the disk library system drive(s) are the same as in a Model 650/A, you can use the same optical disk in either of the two devices.

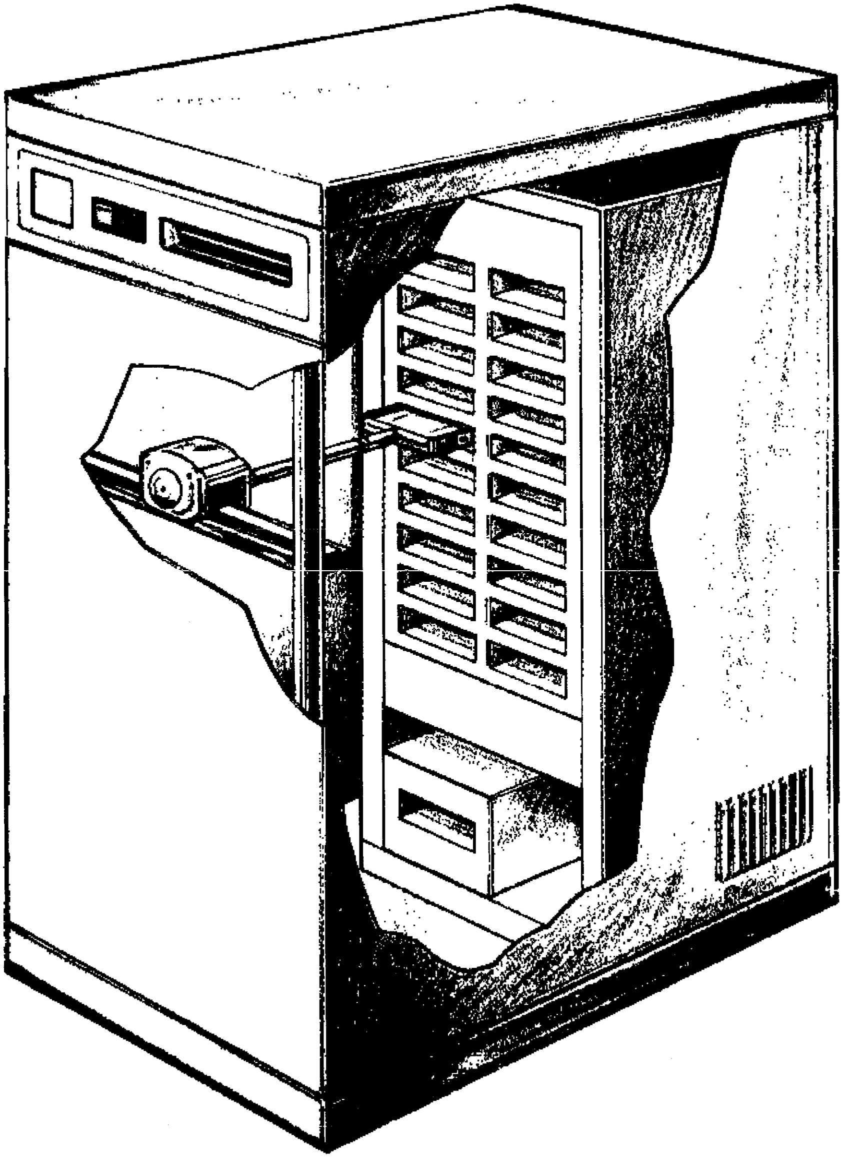


Figure 12-4. Optical Disk Library System Concept

Optical Disk Library System Guidelines

Each of the 64 surfaces in the disk library system (32 disks, double-sided) is treated like a hard disk. A surface can hold a file system, can be stored to and rewritten like a hard disk, has a device file associated with it, and can be accessed with the same HP-UX commands as a magnetic disk.

While each surface on the disk is treated like a magnetic disk, the disk library system is not a hard disk replacement. With this in mind, follow these guidelines when using an optical disk library system:

- Boot Device: Do not use any surface in the optical disk library system as the boot device. It is not supported as a boot device.
- Virtual Memory Swap Space: Do not put swap space on any surface in the disk library system. Swap space is not supported.
- Directories on the Search Path: Do not put directories that are on optical disk surfaces in your search path (defined by the PATH environment variables). This could cause performance delays.
- Mounting Surfaces:
  - Mount no more than one surface in any branch of the file tree.

    Although you can mount surfaces anywhere in the file system, for best performance no more than one surface should be mounted in any branch of the file tree. Additional disk exchanges will result when traversing these paths.
  - Minimize the number of surfaces mounted for writing.

    You can have as many read/write surfaces as you want; however you should minimize the number of surfaces mounted read/write.

    If the system's power fails, all write-mounted surfaces will require a file system check (`fsck`). The time required to check the file system if you have many optical disk surfaces mounted could be excessive.
  - Do not put surfaces in `/etc/checklist`.

    This slows down the booting process considerably since each surface must be exchanged to be mounted and possibly file system checked. A better strategy is to write a script to mount surfaces after the system is booted.

This way, activity on the rest of the system can proceed while the surfaces are being mounted.

- Copying Files:

12

  - Don't copy files straight from side A to side B of the same disk. Doing so requires many exchanges. Rather, copy the file intermediately to the hard disk and then from the hard disk to the other optical surface.
  - If you have a one-drive Disk Library System, don't copy straight from one optical disk surface to another.

Terminals and Modems

The following sections explain some conceptual information on terminals, and configuring terminals into your system. The number of terminals and ports you can add depends on the model and configuration of your computer system.

Use the procedures in the *System Administration Tasks* manual to set up terminals.

Terminal Configuration

The *Installing Peripherals* supplied with your computer discusses the hardware aspects of connecting a terminal to your system. This section offers the software configuration information you need.

After connecting the hardware, terminals must be configured so they can communicate with HP-UX. If a particular configuration option is not available on your HP terminal, the option is already properly chosen (as a default value) by the terminal.

Use the *Installing Peripherals* to determine the typical terminal and data communications parameters. The manual supplied with the terminal describes how to use the function keys to configure the terminal. Generally, you press a key that chooses the "terminal configuration" option and alter the appropriate fields by answering prompts from the terminal's configuration program.

Except when using the terminal as a system console (discussed below), you may use any **baud rate** that the terminal can handle. The baud rate setting on the terminal **must** match the baud rate parameter in the **getty** command located in the terminal's entry in the `/etc/inittab` file as discussed below. Otherwise, it must reference a `gettydefs` entry that cycles the baud rate to select the correct one.

If you are using the terminal as the system console, set the terminal's baud rate at 9600. After the system is installed and running, you may change some of the configuration parameters to suit your own needs. Further information is provided in the next section ("Special Considerations for Terminals") and in *getty*(1M), *gettydef*(4), and *inittab*(4) in the *HP-UX Reference*.

12

Special Considerations for Terminals

For terminals, as well as modems, printers, or X.25 packet assembler/disassembler (PAD) that connect via RS-232 cable, consider the following:

- Determine if a MUX card needs to be added or moved. On Series 800 systems using CIO/Mid-bus architecture, several MUX cards are already configured in the CIO slots. To change the configuration, you need to reconfigure the system using `uxgen`.
- Select which port to use for the device. On Series 800 systems attach the device to an available port on the terminal port distributor (TPD, also known as the junction panel).

/etc/inittab Entries for Terminals

To add a terminal to the system, you must perform the steps described in *System Administration Tasks* manual. Part of this procedure involves adding entries to the `/etc/inittab` file. This file displays a login prompt on configured terminals. `/etc/inittab` is described in Chapter 2, "Constructing an HP-UX System" and Chapter 4, "Controlling Access to the System" of the *System Administration Tasks* manual. This section discusses entries specific to terminals.

Most terminal entries in `/etc/inittab` have the form:

> *id*:*run-level*:*action*:`/etc/getty -t`*xxx* *specialfilename* *N* `# comment field`

Typical values for the *id*, *run-level*, and *action* fields are:

- *id* is a unique string up to 4 characters in length identifying the entry
- *run-level* is 2 (the default multi-user run-level)
- *action* is `respawn`; this indicates that the specified command (`getty` in this case) is to be re-invoked once the process terminates (typically, when the user logs off).

The *process* field contains the `/etc/getty` command for a terminal. It can be followed by up to three parameters:

`-t`*xxx* is the optional time-out option for use with modems.

specialfilename is the filename of the terminal's or modem's character special file (e.g., `tty0p4`), but *not* the complete pathname (e.g., `/dev/tty0p4`). The named file must reside in the `/dev` directory.

12

N specifies a speed indicator for `getty`; a value of 9600 is common for **hardwired** (9600 baud terminal) lines, a value of 1200 is common for dial-up (300/1200 baud modem) lines. Graphics displays ignore the speed indicator.

The *N* parameter indexes into the `/etc/gettydefs` file. It is possible to set up a terminal to cycle through 22 different baud rates by formatting the `gettydefs` file properly. This file supplies information on the speed and terminal settings and what the login prompt should be. If the user enters a "break" on the terminal keyboard, it indicates that the terminal speed is incorrect, which causes `getty` to use the next entry from the `gettydefs` file.

(For details on `getty` parameters, see *getty*(1M) in the *HP-UX Reference*.)

On a multiuser system, be certain to set up `/etc/inittab` terminal entries *for each terminal* connected to the system. For example, to add a terminal on `/dev/tty0p4` the `etc/inittab` could be:

```
p4:2:respawn:/etc/getty tty0p4 9600 #terminal at rob's desk
```

An *id* of `p4` was chosen to correspond to the last two digits of the terminal's special file, although any *id* up to 4 characters long would have sufficed. In general, though, try to use *id*s that enhance the entry's readability. The *run-level* is 2 (multi-user, default run-level), and the *action* field is `respawn`, causing `getty` to restart and the "login:" prompt to be displayed again after a user logs out.

12

HP-IB Guidelines

Depending on your computer system, you can connect to an HP-IB bus through one of several I/O paths, as summarized in the following table:

Table 12-1. HP-IB Connectivity

Series	Type	I/O Path
300	Internal External	Human Interface Board DIO I/II
400	External	EISA
700	External	EISA
800	External	HP-PB CIO

All external HP-IB connections require that your system have HP-IB I/O hardware (optional on some systems). In all cases, you will need appropriate device drivers configured into the kernel and device files created. Each device attached to the bus must have a unique HP-IB address. See the *Installing Peripherals* for further information.

When connecting multiple HP-IB devices, daisy-chain configurations are more reliable than multiple devices connected at a single point.

Cabling requirements affect HP-IB operating speed. Consult the *Installing Peripherals* and documentation shipped with your peripheral for guidelines.

When using HP-IB and other multiple-access external buses, consider what priority the peripherals should have for accessing system resources. HP-IB connections provide either standard-speed or high-speed data transmission, depending on the slowest peripheral connected to the bus. (The exception to this is the standard-speed internal HP-IB connection on the Series 300, which transmits data only at standard speed.)

For example, a high-speed HP-IB might be used to advantage for accessing the system disk containing root and swap, while standard-speed HP-IB might be configured with a cartridge-tape drive, which takes longer to respond and might interfere with higher-speed devices.

The system printer can be installed on a standard-speed HP-IB interface, with bus address of 1. Install the printer separate from the system root device. (The printer can be on high-speed HP-IB, as long as there are no high-speed devices on the same card.) Printers and plotters cannot be on the same HP-IB interface as disks.

On the Series 300 and 400, the system root device (hard disk) is typically located at select code 14, bus address 0 on a HP 98625B or HP 98262A high-speed disk interface card.

On a Series 300, the internal HP-IB is pre-set at select code 7. You can change this select code in the Boot ROM configuration mode. The Boot ROM configuration mode is discussed in the Appendices of the *Installing Peripherals* manuals. On the Series 300, 400, and 800, consider the following factors:

- An HP 7974 or HP 7978 nine-track tape drive should be placed on a high-speed HP-IB interface, if possible. You can also use that same HP-IB for the root device.
- Plotters and the HP 9111 graphics tablet should be placed on separate standard-speed HP-IB interfaces when possible. Typical HP-IB addresses are 5 and 6 for plotters and graphics tablets, respectively.
- Avoid putting flexible disk drives and cartridge tape drives on the same HP-IB interface as the root device, except as noted above.

HP-IB Electrical Guidelines

- Always turn on the HP-IB device before turning on the computer system. Turn off the computer system before turning off the HP-IB device.
- Never turn on or off an HP-IB device while connected to a computer system that is turned on. This could corrupt data on the HP-IB bus.
- All devices attached on an active HP-IB subsystem should have their power switched on. Devices with power switched off load the bus to excess.
- Never connect or disconnect an HP-IB device while the computer system is running or the bus is active.
- Before resetting switches on any HP-IB device, turn off the power to the device.

SCSI Device Guidelines

A SCSI bus may use either single-ended ("unbalanced") or differential ("balanced") electrical signal lines to transfer data. Do not mix single-ended and differential devices and buses; they are electrically incompatible.

The SCSI interface can support multiple SCSI devices simultaneously. A bus terminator must be attached to the end of the chain of devices, or the bus will not operate. The terminator must be compatible with either the single-ended or differential SCSI bus.

See the system configuration guides, *Installing Peripherals*, and the documentation shipped with your peripheral device for specific information about which SCSI devices are supported by your system, whether your device is single-ended or differential, cabling requirements and terminating considerations.

Consider also the following list of SCSI guidelines:

- Use of third party peripherals is at user's risk and is not supported by Hewlett-Packard.
- Never attach a single-ended SCSI device to a differential SCSI bus, or vice versa.
- Each end of a SCSI bus must be terminated. Terminator resistors are typcially installed in the host adapter; thus, the host adapter must be on one end of the SCSI bus. This terminator provides matching impedance on the bus circuits. Without the terminator the bus will not work.
- The last SCSI device in the chain, even if it is the only one, must have a terminator installed to its second connector. This terminator provides matching impedance on the bus circuits. Without the terminator the bus will not work.
- Make sure there are no unterminated cables (that is, all cables are attached to a device at both ends).
- All devices must be connected to a common (single point) system reference ground. The system ground must be isolated from other electrical devices such as copying machines, arc welders and air conditioners. HP supplied cables supply correct grounding.

- Keep all devices powered on during and after system boot-up.
- Do *not* connect or disconnect any SCSI devices while the system is powered on.
- Do *not* turn power on or off to any SCSI device while connected to a powered-up system. This could result in data corruption or a system panic, which in turn might lead to file system corruption.
- All devices must have an unique bus address between 0 and 6. The host adapter's SCSI bus address is typically 7.
- SCSI bus address 7 is reserved for the SCSI interface. Do not use it for SCSI device bus addresses.
- The BootROM searches from bus address 6 to 0. Your root disk should have a higher bus address that any other device on the bus. It is recommended that the root disk be set to SCSI address 6.
- If you need to change the bus ID on a SCSI device, perform the task in the following sequence:
  1. Shut down your system and turn power off.
  2. Turn off the device.
  3. Change the bus ID on the device.
  4. Turn on the device. Power on all SCSI peripherals and make sure they have time to complete their selftest before powering on the SPU (System Processor Unit).
  5. Reconnect the power cord to your system.
  6. Turn on your system.
- Because SCSI cable impedance and construction can have a significant effect on signal quality, only HP cables are recommended.
- Ensure that the total cable length (including external and internal cables) does not exceed SCSI bus limits. For single-ended SCSI, the limit is 6 meters. For differential SCSI, the limit is 25 meters.

  The length of the SCSI bus should be kept as short as possible. However, do not use cables less than 0.5m in length. Refer to your system configuration guides for internal cable length of supported peripherals.

13

Networking

Networking enables you to transfer data among different systems. Networks can be simple (for example, linking just a few machines of the same type) or powerful (having machines from different vendors with different operating systems on each).

Most networking capabilities require that you have optional networking services installed. What services you have depend on what options you used when ordering HP-UX.

This chapter summarizes briefly:

- Network architecture and the OSI model
- HP-UX networking software
- HP-UX networking documentation

Network Architecture and the OSI Model

A network architecture is a structured, modular design for networks. HP's network architecture conforms to the reference model of Open Systems Interconnection (OSI), a network architecture mode developed by the International Standards Organization (ISO). NS, ARPA, and NFS Services are implemented based on the OSI model.

13

In the OSI model, connectivity is divided among seven logically distinct modules called layers, each of which performs a specific data communication function. Interfaces between layers allow communication with adjacent layers or with a peer layer on a remote computer.

Table 13-1 shows each layer and its function.

Table 13-1. OSI Model and Layers

Layer and Name	Layer Function
7—Application	Provides user services, such as file transfer and terminal access to the network.
6—Presentation	Manipulates user data such as in data compression.
5—Session	User's interface into the network. Establishes and manages connections between users.
4—Transport	Responsible for data integrity, ensures that data arrives at a remotely networked computer without errors, controls flow, retransmits corrupt data, suppresses duplicated data.
3—Network	Routes messages from one node to another.
2—Data Link	Packs raw bits for transmission, detects transmission errors, controls access to the media.
1—Physical	Defines the electronic characteristics of the transmission media.

HP-UX Networking

Networking software corresponds to specific layers of the OSI model and can be thought of in three broader categories:

- links, corresponding to layers 1 and 2.
- transports, corresponding to layers 3 through 6.
- services, corresponding to layer 7.

Table 13-2 shows the relationships of links and transports to HP-supported networking services. For example, the LAN link enables access to Network Services (NS) through the TCP/IP transport layer or to FTAM through the OTS layer.

Table 13-2. HP 9000 Networking Products

Links	Transports	Services
LAN	TCP/IP	NS, ARPA, NFS
	OTS	X.400, FTAM
X.25	TCP/IP	NS, ARPA
	OTS	X.400, FTAM
802.4	HP OSI Express	FTAM, MMS

Some of these components are bundled into integrated products. For example, the TCP/IP transport is bundled with LAN and X.25 products; the 802.4 link is included in the HP OSI Express MAP 3.0 product.

Networking Documentation

For a good overview of all networking services, refer to the manual *Networking Overview*. Also, networking reference pages are located in the *HP-UX Reference*. The designation `(3N)` denotes networking library.

As a system administrator, you are most likely to deal with networking issues from the perspective of services. Table 13-3 summarizes the services implemented in HP-UX and manuals available for each element in the preceding table. (Manuals can be ordered through your local HP sales representative.)

Table 13-3. Available Networking Services

Service	Capabilities	Related Documentation
ARPA/ Berkeley Services	Connects HP 9000 to other HP 9000 systems or multivendor systems running ARPA/Berkeley Services from HP or non-HP software supplier. Requires optional ARPA Services software.	■ *Using ARPA Services* ■ *Installing and Administering ARPA Services* ■ *Berkeley IPC Programmer's Guide*
NFS/9000	Allows users on a networked computer to access files from another HP-UX or UNIX system over the network as though those files resided on the client's file system. Requires optional NFS Services software.	■ *Using NFS Services* ■ *Installing and Administering NFS Services* ■ *Programming and Protocols for NFS Services*
UUCP	Enables users to transfer files among HP-UX and other UNIX systems, and to execute commands on remote systems. Computers must have UUCP installed and be connected via modem lines. UUCP utilities reconcile differences in file format between systems. UUCP is provided as part of standard HP-UX.	*UUCP* volume of *HP-UX Concepts and Tutorials*
NS/9000	Provides network file-transfer capability between HP 9000, HP 3000, HP 1000 and multivendor systems running NS, HP's proprietary network implementation. Used to access applications and databases. Requires optional Network Services software.	■ *Using Network Services* ■ *Installing and Administering Network Services* ■ *LLA Programmer's Guide* ■ *NetIPC Programmer's Guide* ■ *NS Cross-System NFT Reference Manual*

Table 13-3. Available Networking Services (continued)

Service	Capabilities	Related Documentation
OTS/9000	Provides networked communication using OSI Transport Services protocols between multivendor systems. HP-UX OSI supports FTAM (File Transfer, Access, and Management service), X.400 (a standard electronic mail service), MMS (Manufacturing Message Specification used for multivendor manufacturing automation applications). Includes programmatic access to the transport layer via the X/Open Transport Interface (XTI). Requires optional OSI Transport Services software.	■ *Installing and Administering HP OSI Transport Services* ■ *HP OSI Express 802.4 Hardware Installation and Configuration Guide* ■ *Installing and Administering HP OSI Express MAP 3.0 Link* ■ *Troubleshooting HP OSI Express MAP 3.0 Link* ■ *Administrator's Reference Manual for HP OSI Express* ■ *XTI Programmer's Guide* ■ *Session Level Programmer's Guide*
X.400/9000	Enables HP 9000 system users to sending and receiving electronic mail messages throughout a multivendor network. Requires optional X.400 software.	*Installing and Administering X.400/9000*
X.25	Connects HP 9000 systems to public and private X.25 wide-area networks (WANs).	■ *X.25 Programmer's Guide* ■ *X.25: The PSN Connection*
LAN/9000	Provides network connection from HP 9000 systems to Ethernet and local area networks. Enables information exchange through NetIPC and Berkeley Sockets, two interprocess communication facilities. Allows implementation of link-level access (LLA) for special-purpose network protocols to access network interface drivers directly. Requires optional LAN/9000 software.	■ *Installing and Administering LAN/9000 Series 300* ■ *Installing and Administering LAN/9000 Series 800* ■ *LLA Programmer's Guide* ■ *Berkeley IPC Programmer's Guide* ■ *NetIPC Programmer's Guide*

13

Table 13-3. Available Networking Services (continued)

Service	Capabilities	Related Documentation
FTAM/9000	Provides multivendor file system interoperability, permitting text and binary file transfer, access to remote files, or management of remote files of any defined data type between HP9000 and diverse OSI end systems, regardless of hardware or operating system. Requires optional FTAM/9000 software.	▪ *HP FTAM/9000* ▪ *HP FTAM/9000 Programming Guide* ▪ *HP FTAM/9000 User's Guide*
MMS/9000	Enables MAP-connected cell controllers to direct control of MAP 3.0 conformant factory-floor devices via program-to-program messaging and binary file transfer services.	▪ *MMS/9000 Reference Manual* ▪ *MMS/9000 Programmer's Guide*

14

System Accounting

HP-UX allows users to share computer resources such as disk space, memory, and the CPU. Some computing environments, such as universities and government contractors, need to keep and report detailed accounting information on the use of these resources. Time-share environments also require detailed records of resource use.

HP-UX is equipped with programs and scripts that tabulate resource usage and generate reports. In addition, you can develop custom scripts that use the available HP-UX accounting tools.

This chapter introduces the principles and tools of system accounting, as indicated in the next section. For accounting procedures, refer to the *System Administration Tasks* manual for your system.

What Is in This Chapter?

HP-UX system accounting allows you to accomplish accounting tasks using many versatile commands. This chapter describes concepts underlying use of these accounting commands in the following sections:

- "Accounting System Planning and Billing" suggests how to approach setting up and using an accounting system.
- "Overview of System Accounting" gives background information useful for using system accounting.
- "System Accounting Set-up Guidelines" summarizes what happens when you set up, enable and disable system accounting.
- "System Accounting Files" shows the directory structure and contains brief definitions of system accounting files.
- "Disk Space Usage Accounting" illustrates the use of the accounting commands that monitor disk space utilization on a per-user basis.
- "Connect Session Accounting" describes the commands that record and report connect session accounting information.
- "Process Accounting" discusses methods for generating per-process accounting data and reports.
- "User Fees" introduces an HP-UX tool for charging fees to users.
- "Accounting Summaries and Reports" surveys the daily and monthly accounting reports that can be generated to monitor system performance and bill users.
- "Disk Quotas" describes a system of tools that enable you to limit disk usage by user or file system.

Accounting System Planning and Billing

As a system administrator, you might be asked to provide data that can be used to bill individuals or groups for their use of computer resources.

To develop an equitable way of sharing costs, you'll need to gather some information:

- how to group people for billing
- what criteria to apply for billing computer resources

Then, with the accounting software described in this chapter, collect appropriate data and apportion costs by use. In this way, heavy and light users are fairly charged.

Grouping Users for Billing

You can group individuals logically for data collection purposes, so that resource costs can be billed to these groups in proportion to the amount of resources they consume. You might do this by using a small database with login name or user ID number associated with the entire group or functional area.

Another possibility is to use the GCOS (personal information) field of `/etc/passwd`, so that the information can be extracted using `grep` or `awk`.

Note that the *chfn*(1) command allows users to change their entries in `/etc/passwd`, so with this approach, you might want to restrict access to `chfn`.

Data collected with HP-UX accounting tools and a grouping technique can be used as input to a more global spreadsheet or charging scheme.

Criteria for Billing Computer Resources

Find out what criteria or costs are key to account for on a given machine. For example, system-administration time for tasks such as backups might be critical in a heavily file-system based environment. In an NFS cluster or file server, you might care more about CPU cycles. Figure out the most critical elements, use the appropriate accountancy tools to derive your data, and apportion charges based on your findings. Thus, you might decide to apportion charges as follows, based on what you perceive as key costs:

100% charges = 30% CPU + 70% disk space usage

Time Considerations

When collecting data by connect-session and process accounting, you can bill usage of computer resources differently during busy or less busy periods. When reporting computer time usage, system accounting distinguishes between prime and non-prime time, which is defined in the `/usr/lib/acct/holidays` file.

Prime vs Non-prime Connect Time. **Prime time** is the time during the day when the computer system is most heavily used—typically from 9:00am to 5:00pm. **Non-prime time** is the remaining time during the day when the system is used less—typically from 5:00pm to 9:00am.

Prime time is in effect only on weekdays (Monday through Friday). Non-prime time is in effect during the weekends (Saturdays and Sundays) and on any
14 holidays specified in the `holidays` file.

You can specify prime and non-prime time on your system by editing the file `/usr/lib/acct/holidays`.

Updating the Holidays File. The file `/usr/lib/acct/holidays` contains the information that System Accounting needs to distinguish between prime and non-prime time. Here is a typical `holidays` file:

```
* Prime/Nonprime Table for HP-UX Accounting System
*
* Curr  Prime   Non-Prime
* Year  Start   Start
*
  1990  0800    1700
*
* Day of        Calendar        Company
* Year          Date            Holiday
*
  1            Jan 1           New Year's Day
 50            Feb 19          Washington's Birthday
103            Apr 13          Spring Holiday
148            May 28          Memorial Day
185            Jul 4           Independence Day
246            Sep 3           Labor Day
326            Nov 22          Thanksgiving Day
327            Nov 23          day after Thanksgiving
358            Dec 24          Christmas Eve
359            Dec 25          Christmas
```

14

Note the following characteristics of the `holidays` file:

- **Comment Lines**—Comment lines begin with an asterisk (*) and can appear anywhere in the file.
- **Year Designation Line**—This line should be the first non-comment line in the file and must appear only once. The line consists of three four-digit numbers representing year, time (in 24-hour format) that prime time starts, and time that non-prime time starts.

  The time field entry `2400` is identical to `0000`.
- **Company Holidays**—Since few people use the computer during holidays, non-prime time applies for the 24 hours of those specified days.

Note As delivered, the `holidays` file contains valid entries for typical prime/non-prime time, and holidays. Edit this file to reflect your organization's requirements.

Collecting Data Strategically

Having determined your billing plan, you can use accounting tools to determine usage, collect statistics, and input data. This information can then be used as input to either the `chargefee` billing command or a more global billing scheme, which might also include variables such as occupancy and depreciation.

Overview of System Accounting

HP-UX commands let you:

- Monitor disk space usage of individual users.
- Record connect-session data (logins/logouts).
- Collect resource-utilization data (such as memory usage, execution times for individual processes.
- Charge fees to specific users and groups.
- Generate summary files and reports used to analyze system performance and bill users for resource consumption.

As described in "Accounting System Planning and Billing," system accounting can be used effectively to quantify and bill for computer system usage. Regardless of complexity, determining those statistics involves two aspects:

- Gathering and displaying data
- Generating reports

14

Figure 14-1 shows the elements that contribute to this process in relationship to one another. In a general sense, the "environment snapshot" refers to values generated by the various commands at the time they are invoked. The content of various files also can be merged for intermediate summaries, and the output of one command is often used as input to other commands.

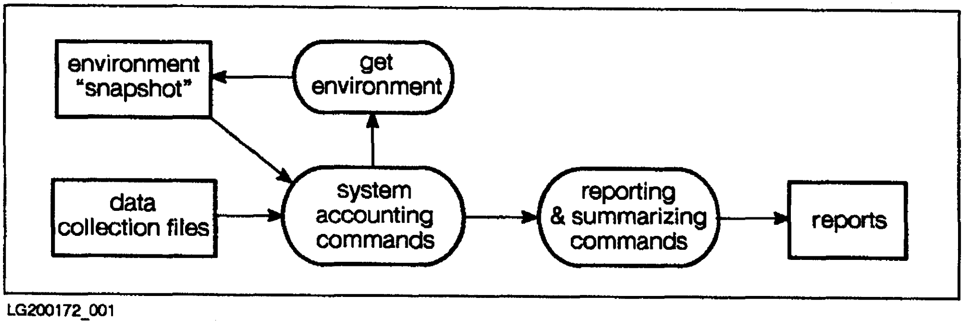


Figure 14-1. System Accounting Overview

Gathering Data

HP-UX System Accounting provides numerous tools for gathering data, which is then used to generate reports and possibly charge fees. Most system accounting focuses on:

- Disk space usage
- Session accounting
- Process accounting

We discuss these categories one by one in subsequent sections of this chapter.

Automating Data Collection

Gathering data can be automated.

14

You can use the `cron` command to automate routine accounting tasks, by setting `cron` to execute various accounting commands at regularly scheduled intervals. For example,

- Run the `ckpacct` command every hour to ensure that the process accounting file `pacct` does not become too large. If `pacct` grows past a set maximum number of blocks, `turnacct switch` is invoked, which creates a new `pacct` file. (Other conditions may also limit the size of the process accounting file or turn process accounting off; for more details, refer to the discussion of `ckpacct` in the "Process Accounting" section of this chapter.) The advantage of having several smaller `pacct` files is that `runacct` can be restarted faster if a failure occurs while processing these records.
- Execute `runacct` each night, to process active fee files, as well as process-accounting, connect-session, and disk-usage accounting files. The `runacct` command produces command and resource-usage summaries by login name.

  A report of the previous day's accounting information can be created by running `prdaily`, once `runacct` has been executed.
- The `chargefee` program can be used to charge fees to users. It adds records to the file `fee`. These records are processed during the next execution of `runacct` and merged in with total accounting records.

These suggestions will make more sense once you have read through the chapter and decided upon your overall approach.

Generating Reports

Raw data either represents a snapshot of the state of the computing system or is generated by system-accounting tools and collected in files from which reports can be generated. Flow charts in subsequent sections show how data is synthesized into assessible form, called total accounting records.

Total Accounting Records

These records, created by various accounting commands, contain summary accounting information for individual users. The format of total accounting records conforms to a structure `tacct.h`, explained in *acct*(4) of the *HP-UX Reference*.

Here are some examples of information contained in these records:

- ID and user name of the target user
- Total number of processes a user has begun during a given accounting period
- Fees for special services rendered to the user

These records provide the basic information for many reports generated by system accounting. Commands covered in later sections of this chapter show how these records are created and used by system accounting.

Summary Reports

The data tabulated in total accounting records can provide insight on specific areas. They can also be combined to produce comprehensive summaries of computer resource usage. Several tools for summarizing and reporting accounting information are discussed later in this chapter.

System Accounting Set-up Guidelines

Since not all computing environments require accounting capabilities, and accounting does impact performance, HP-UX System Accounting is provided as a selectable fileset. Before you can generate accounting data and reports, you must enable the accounting software.

To set up system accounting, you perform three steps:

1. Update `/etc/rc` to automatically start System Accounting when the system switches to multi-user mode.
2. Create `crontab` entries to automate creation of accounting data.
3. Set PATH to include the directories where accounting commands reside.

These steps are discussed in detail in the *System Administration Tasks* manual for your system.

14

Administering System Accounting in a Cluster (Series 300/400/700 only)

System accounting in a clustered environment is straightforward, as long as you realize where data is stored and processes execute. Some issues need special consideration; for example, on an HP-UX cluster:

- You must enable or disable accounting on each cluster client.
- Some tasks, such as disk-space accounting, provide useful data only when administered from the root server.

Note Throughout this chapter, we note cluster-specific details. For further information on clusters, see *Managing Clusters of HP 9000 Computers: Sharing the HP-UX File System*.

How System Accounting is Turned On

When HP-UX is switched into multi-user mode, the system initialization shell script `rc` executes the command `startup`, which starts system accounting by performing the following functions:

- Calls `acctwtmp` to add an initial record to `/etc/wtmp`. This record is marked by storing "acctg on" in the device name field of the `/etc/wtmp` record.
- Turns on process accounting via `turnacct on`, which executes `accton` with the filename argument `/usr/adm/pacct`. This directs the kernel to collect process-accounting data in the file `/usr/adm/pacct`. (Additional information on turning on process accounting is found later in this chapter.)
- Removes work files left in the `/usr/adm/acct/sum` directory by `runacct`.

14

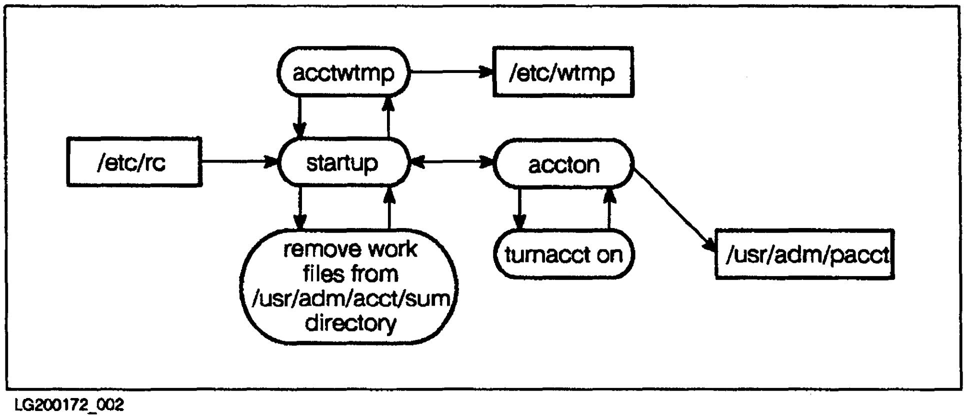


Figure 14-2. How startup Enables System Accounting

How System Accounting is Turned Off

When a system that runs accounting is turned off using `shutdown`, the `shutacct` command is executed. The purpose of `shutacct` is to stop System Accounting by performing the following functions:

- Writes a termination record to `/etc/wtmp` by calling the command `acctwtmp`, which writes "acctg off" in the device name field.
- Turns process accounting off by calling `turnacct off`. (Additional information on turning off process accounting is found later in this chapter.)

Logging In for System Accounting

14

To maintain the integrity of accounting data files, we recommend that you establish a separate accounting user account, *with password*, for use when performing all accounting functions.

The login name for system accounting is `adm`; the user ID for `adm` is 4. The `adm` login is a member of the *group* `adm`, whose group ID is also 4. The home directory for the `adm` login is `/usr/adm`.

System Accounting Files

System Accounting uses a multi-level directory structure to organize its many accounting files, some of which are permanent and others temporary. Each directory in this structure stores related groups of files, commands, or other directories.

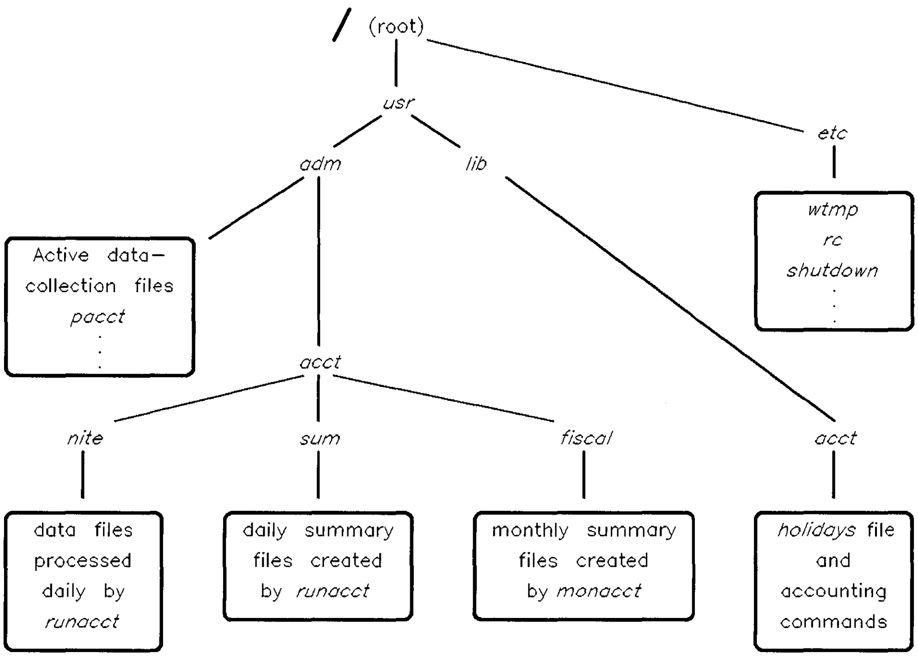


Figure 14-3. System Accounting Directory Structure

14

- /usr/adm—contains all active data-collection files, such as pacct and fee.
- /usr/adm/acct—contains the nite, sum, and fiscal directories described below.
- /usr/adm/acct/nite—stores data files that are processed daily by runacct.
- /usr/adm/acct/sum—cumulative summary files updated by runacct are kept here.
- /usr/adm/acct/fiscal—periodic (monthly) summary files created by monacct are stored here.
- /usr/lib/acct—System Accounting commands reside here.
- /etc—contains wtmp, and shell scripts rc and shutdown.

Files in the /usr/adm Directory

Note that in an HP-UX cluster (Series 300/400/700 only), the /usr/adm directory is a context-dependent file (CDF).

Filename	Contents
dtmp	Output from the acctdusg or diskusg programs.
fee	Output from the chargefee command (ASCII total accounting records).
pacct	The current active process accounting file.
pacct?	Prior process-accounting files switched via turnacct switch.

Files in the /usr/adm/acct/nite Directory

Filename	Contents
`active`	Used by `runacct` to record progress. It contains warning and error messages. `active`*mmdd* is the same as `active` after `runacct` detects an error.
`ctacct.`*mmdd*	Total accounting records created from connect session accounting where *mmdd* is the month and day the file was created.
`ctmp`	Output of `acctcon1`—connect session records.
`daycms`	ASCII daily command summary used by `prdaily`.
`daytacct`	Total accounting records for current day.
`disktacct`	Total accounting records created by the `dodisk` command.
`fd2log`	Diagnostic output from the execution of `runacct` (refer to `crontab` entry).
`lastdate`	The last day `runacct` was executed, in `date|#`*+%m%d* format. (Refer to *date*(1) in the *HP-UX Reference* for a description of *+%m%d* date format.)
`lock` and `lock1`	Used to control serial use of `runacct`.
`lineuse`	Terminal (`tty`) line usage report used by `prdaily`.
`log`	Diagnostics output from `acctcon1`.
`log`*mmdd*	Same as `log` after `runacct` detects an error.
`reboots`	Contains beginning and ending dates from `wtmp`, and a listing of reboots.
`statefile`	Used to record the current state being executed by `runacct`.
`tmpwtmp`	`wtmp` file, corrected by `wtmpfix`.
`wtmperror`	Error messages, if any, from `wtmpfix`.
`wtmperror`*mmdd*	Same as `wtmperror` after `runacct` detects an error.
`wtmp.`*mmdd*	The previous day's `wtmp` file.

Files in the /usr/adm/acct/sum Directory

Filename	Contents
`cms`	Total command summary file for current month in internal summary format.
`cmsprev`	Command summary file without latest update.
`daycms`	Command summary file for previous day in internal summary format.
`loginlog`	Shows the last login date for each user.
`rpt`*mmdd*	Daily accounting report for date *mmdd*.
`tacct`	Cumulative total accounting file for current month.
`tacctprev`	Same as **tacct** without latest update.
`tacct`*mmdd*	Total accounting file for date *mmdd*.
`wtmp.`*mmdd*	Saved copy of wtmp file for date *mmdd*. Removed after reboot.

14

Files in the /usr/adm/acct/fiscal Directory

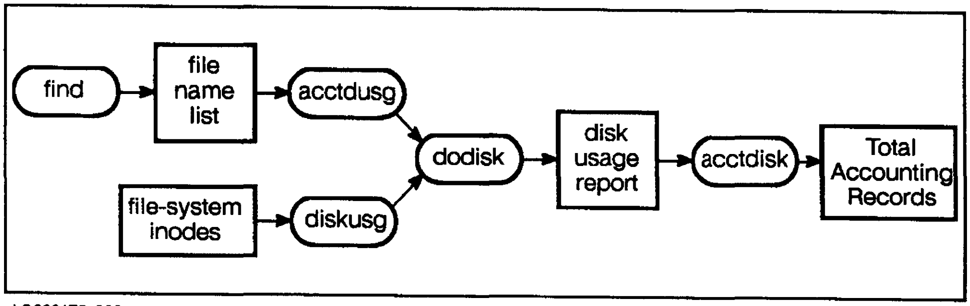
Filename	Contents
`cms`*?*	Total command summary for month *?* in internal summary format.
`fiscrpt`*?*	Report similar to **prdaily** for the month *?*.
`tacct`*?*	Total accounting file for the month *?*.

Disk-space Usage Accounting

System accounting provides the means to monitor disk space used by individual users. Before reading this discussion, you might want to review the "HFS File System" chapter of this manual. Also, if you are interested in setting limits on disk-space usage, refer to the section at the end of this chapter, entitled "Disk Quotas."

Disk-usage commands perform three basic functions, which are illustrated in the figure below:

- tabulate disk usage by file system
- report disk usage (in blocks) for individual users
- summarize disk usage in a form that can be merged with other accounting records, to produce reports by commands such as `prtacct` or `runacct`.



LG200172_003

Figure 14-4. Disk-space Usage Accounting

Reporting Disk Space Usage

Two commands—`acctdusg` and `diskusg`—report disk usage by showing the number of disk blocks individual users are consuming. Each command has slightly different options and differs in how it produces accounting information.

Note that neither `acctdusg` nor `diskusg` produce any output on a cluster client that does not have a file-system disk attached to it. For information on working in a clustered environment (supported on Series 300/400/700 only), see *Managing Clusters of HP 9000 Computers: Sharing the HP-UX File System.*

Computing Disk Resource Consumption by Login—acctdusg

`Acctdusg` is a script that takes its input from a list of path names created by the `find` command. For each file in the list, `acctdusg` identifies the owner of the file, computes the number of blocks allocated to the file, and adds this amount to a running total for the file's owner. `acctdusg` displays the information accumulated for each user: user ID, login name of the user, and number of blocks used.

The syntax for `acctdusg` is:

`acctdusg` [`-u` *no_owners*] [`-p` *p_file*]

`Acctdusg` has two useful options:

`-u` *no_owners* — If `-u` is given, path names of any files without apparent owners are written into the file *no_owners*. This option could potentially find users who are trying to avoid disk charges.

14

A shell script called `grpdusg`, described in *System Administration Tasks*, displays disk accounting information for users in a given group and uses the `-u` option.

`-p` *p_file* — The password file `/etc/passwd` is the default file used by `acctdusg` to determine ownership of files. If the `-p` option is used, then `acctdusg` will use *p_file* instead. This option is not needed if your password file is `/etc/passwd`.

For more information about `acctdusg`, refer to *acct*(1M) in the *HP-UX Reference*

Generating Disk Accounting Data by User ID—diskusg

`Diskusg` generates disk usage information by examining the *inodes* of the file systems associated with a disk, to create its output. (Refer to *inode*(4) in the *HP-UX Reference* and the "File Systems" chapter of this manual for more information on inodes.)

This command reports disk usage information in the same format as `acctdusg`—user ID, login name of the user, and total disk blocks used, but because it reads the code directly, `diskusg` is faster and more accurate than `acctdusg`.

The output of **diskusg** is normally used by **acctdisk** to create disk total accounting records. In addition, **diskusg** is normally called by **dodisk**.

The syntax of the **diskusg** command is:

diskusg [*options*] [*device special files*]

The following options can be used with **diskusg**:

- `-s` — Enables **diskusg** to generate reports from existing **diskusg** output and streamline the output by combining information about a single user found in multiple reports. This option is used to merge data from separate files, each containing the output from using **diskusg** on different devices.
- `-v` — Useful for finding users who are trying to avoid disk space accounting charges. When this option is specified, **diskusg** writes records to **stderr** (standard error output) showing the device file name, inode number, and user ID of files that have no apparent owner.
- `-i` *fnmlist* — Causes **diskusg** to ignore the data on those file systems whose file system name is in *fnmlist*. *fnmlist* is a list of file systems separated by commas or enclosed within quotes.
- `-p` *p_file* — This is the same as the `-p` option of `acctdusg`.
- `-u` *u_file* — This option produces identical output as the `-v` option, but `-v` writes its output to `stderr`, whereas this option writes its output to the file *u_file*.

14

Comparing acctdusg and diskusg

Differences between **diskusg** and **acctdusg** are best illustrated by comparing their results. In the two examples that follow, disk usage information was generated for all users whose files reside on the disk whose device file is **/dev/dsk/0s0**.

In the example using **acctdusg**, the command was executed from the mount-point of the file system.

```
$ find /  -hidden -print | acctdusg
00350   fred            11
00351   bill            30
00352   mike            17
00353   sarah           13
00354   molly           18
00000   root             3
00004   adm             36
00001   bin           2434
```

When using **diskusg**, the device file of the disk is specified.

```
$ diskusg /dev/dsk/0s0
0       root    10616
1       bin     778
4       adm     96
350     fred    14
351     bill    32
352     mike    20
353     sarah   16
354     molly   22
355     julie   2
501     guest   2
```

From these examples, you can see:

- **Acctdusg** places leading zeros on user IDs; **diskusg** does not.
- **Acctdusg** counts files *only* under each user's $HOME directory. Files owned by users outside their home directory hierarchy (for example, files in the /tmp directory) are counted as files without owner.
- Two extra users—**julie** and **guest**—show up in the **diskusg** output. This occurs because the directories of these two users are empty; therefore, no disk usage totals are generated by **acctdusg**. However, **diskusg** looks at inodes and sees that **julie** and **guest** are actually using two blocks for the directories themselves.
- If two or more users have links to a particular file, **acctdusg** divides disk space usage for the file between each user. For example, if three users have

links to a 300-block file called `report.dat`, each user is charged for 100 blocks of this file.

Creating Total Accounting Records

Both `acctdusg` and `diskusg` produce intermediate information that can be redirected into commands that produce total accounting records.

Two commands are used to create total accounting records—`acctdisk`, and `dodisk`.

`acctdisk`
: `Acctdisk` uses standard input from `acctdusg` and `diskusg` and produces disk total accounting records that can then be input to `prtacct` or `runacct` to produce summary reports. `Acctdisk` has no options and is typically invoked by `dodisk`.

`dodisk`
: `Dodisk` produces total accounting records by using the `diskusg`, `acctdusg`, and `acctdisk` commands. `Dodisk` is normally invoked by `cron` to create disk total accounting records for daily usage by System Accounting. The syntax for `dodisk` is:

  `dodisk` [`-o`] [*files* ...]

  In the default case, `dodisk` creates disk total accounting records on the special files whose names are stored in `/etc/checklist`; the special file names are supplied as input to `diskusg`, which pipes its output to `acctdisk`, which in turn creates total accounting records.

  If the `-o` option is used, `dodisk` creates total accounting records more slowly by using `acctdusg` instead of `diskusg`.

  If *files* are used, disk accounting is limited to these file systems only. When the `-o` option is used, *files* should be mount points of mounted file systems; if omitted, *files* should be the device file names of the disk.

  `Dodisk` is one of several shell scripts described on *acctsh*(1M) in the *HP-UX Reference*.

14

Other Disk-related Commands

Besides the accounting commands described in this section, you can get valuable information using two other HP-UX commands:

emph|du|(1) Summarizes disk usage by giving the number of 512-byte blocks contained in files and directories.

emph|bdf|(1) Reports the number of free disk blocks on a specified file system.

A Security Note

It is possible for malicious users to defeat disk-space accounting by assigning their files to other users with *chown*(2) or *chown*(1), which by default, all users can execute. If necessary, restrict access to these commands for some or
14 all users with the `setprivgrp(1M)` command. To restrict `chown` use to the superuser, add the following line to `/etc/rc`:

```
setprivgrp -n CHOWN
```

To permit one or more groups of users to use `chown`, add a line for each group to `/etc/rc`, similar to the following:

setprivgrp *group_name* CHOWN

Bear in mind, however, that restricting access to `chown` might cause problems when running various commands or applications.

Accounting for Disk-space Usage in a Cluster (Series 300/400/700 only)

All cluster nodes in an HP-UX cluster typically share the section of `/etc/rc` that sets privileged groups. This arrangement enables all nodes to behave identically toward privileged groups. However, if you wish, you can make `/etc/rc` a context-dependent file, with different contents for each cluster node. See *Managing Clusters of HP 9000 Computers: Sharing the HP-UX File System* for more information.

Connect-session Accounting

Connect-session accounting is used to record data about when a user logs into a system. This is the form of accounting used by time-share systems and electronic bulletin boards to charge subscribers only for the time spent using system resources, that is, the user's "connect time."

In HP-UX, the programs `login` and `init` record connect sessions by writing records into `/etc/wtmp`. The `/etc/wtmp` file is key to connect-session accounting, as you can see in Figure 14-5. System accounting commands can display or fix this file, and can produce total accounting records for this file.

14

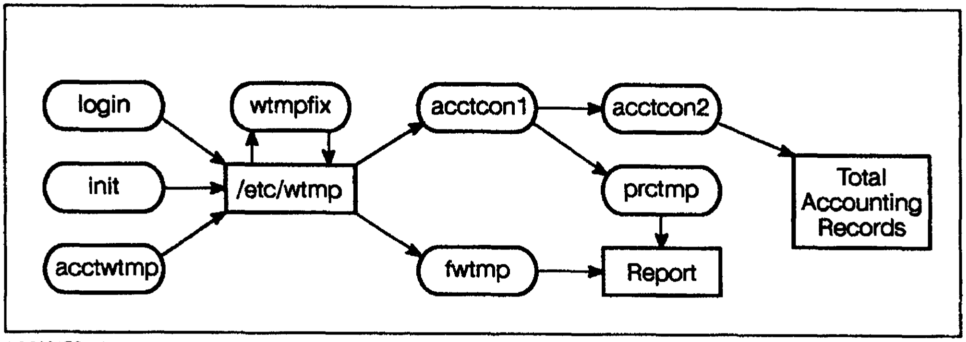


Figure 14-5. Connect-session Accounting

The Key Connect-session Accounting File—/etc/wtmp

The `/etc/wtmp` file is one of three—`/etc/wtmp`, `/etc/utmp`, and `/etc/btmp`—that hold data for such commands as *last*(1), *who*(1), *write*(1), and *login*(1); they are defined by the `utmp.h` header file. Refer to *utmp*(4) in the *HP-UX Reference* for detailed specifications.

Connect-session accounting commands allow you to write records to `/etc/wtmp` and to display and manipulate `/etc/wtmp` data, which cannot be read directly. You can also create total accounting records from `/etc/wtmp`.

Records in `/etc/wtmp` contain the following information:

- the terminal name on which the connect session occurred
- the login name of the user
- the current time/date at login or logout
- other status information

Connect-session Accounting Commands

Six commands are used to account for connect sessions:

- `acctwtmp`—writes records to `/etc/wtmp`.
- `fwtmp`—displays the information contained in `/etc/wtmp`.
- `wtmpfix`—fixes connect-session records that span date changes (refer to *date*(1) in *HP-UX Reference*.). Also validates `login` names in connect session records.
- `acctcon1`—summarizes `/etc/wtmp` in ASCII readable format, producing one line per connect session.
- `acctcon2`—takes input of the format produced by `acctcon1` and produces total accounting records as output.
- `prctmp`—displays the session record file, called by default, `/usr/adm/acct/nite/ctmp`.

Writing Records to wtmp - acctwtmp

The command `acctwtmp` allows you to write records to `/etc/wtmp` for any reason. `Acctwtmp` is typically invoked by `startup` and `shutacct` to record when system accounting is turned on and off, respectively.

The syntax of the command is:

acctwtmp "*reason*"

where *reason* is a very brief comment string in which to describe why you are writing the record to `/etc/wtmp`. Note that `acctwtmp` does not directly write records to `/etc/wtmp`: it writes a record containing the terminal name, current time, and reason string to standard output. To actually write the record to

/etc/wtmp, you must append the output from acctwtmp to the /etc/wtmp file as follows:

```
acctwtmp "reason" >>/etc/wtmp
```

The *reason* string may be any combination of letters, numbers, spaces, and the dollar sign ($), but may not exceed 11 characters in length. (*reason* must be enclosed in double quotes as shown.)

For example, you might want to add a terse remark about why you needed to reboot, such as "out of swap" or "reconfigure".

Refer to *acct*(1M) in the *HP-UX Reference* for specifications on acctwtmp.

Displaying Connect Session Records - fwtmp

To display the contents of /etc/wtmp, you can use the command fwtmp. When no options are used, fwtmp uses standard input records of the format contained in /etc/wtmp; it writes to standard output the ASCII readable equivalent of the input records. 14

The output of fwtmp can be edited, using an HP-UX editor such as vi, and then rewritten to /etc/wtmp using special fwtmp options described below.

The syntax of fwtmp is:

```
fwtmp [-ic]
```

If no option is specified for the fwtmp command, input is in binary format and can be converted to ASCII readable format. Output results differ, depending on whether you use -i, -c, or -ic:

Option	Description
-ic	Input is in ASCII readable form and is to be converted to binary form. This is essentially the opposite of using fwtmp without any options.
-i	Both input and output are in ASCII readable format. This is the same as performing an ASCII to ASCII copy.
-c	Both input and output are in binary format—a binary to binary copy.

The following example shows the output produced by **fwtmp**:

```
$ fwtmp < /etc/wtmp
                    system boot    0  2 0000 0000 479472540 Mar 12 03:49:00 1985
root          co console           0  7 0000 0000 479475173 Mar 12 04:32:53 1985
                    acctg on       0  9 0000 0000 479493135 Mar 12 09:32:15 1985
mike          a1 ttya1           352  7 0000 0000 479493590 Mar 12 09:40:00 1985
mike          a1 ttya1           352  8 0011 0000 479496000 Mar 12 10:20:00 1985
sarah         07 tty07           353  7 0000 0000 479518335 Mar 12 16:32:15 1985
bill          10 tty10           351  7 0000 0000 479521475 Mar 12 17:24:35 1985
sarah         07 tty07           353  8 0011 0000 479522478 Mar 12 17:41:18 1985
bill          10 tty10           351  8 0011 0000 479526487 Mar 12 18:48:07 1985
              co console           0  8 0011 0000 479526488 Mar 12 18:48:08 1985
                    acctg off      0  9 0000 0000 479526493 Mar 12 18:48:13 1985
                    system boot    0  2 0000 0000 479389800 Mar 12 05:00:00 1985
```

The output contains the following information:

- The login name of the user who logged in or out.
- **/etc/inittab** ID (this is usually the number of the line on which the connect session took place).
- Name of the device on which the connect session occurred.
- Process ID of the user who logged in or out.
- Entry type. This field contains information on the type of record—for example, it shows whether the record is a login record (entry type=7), logout record (entry type=8), or if the record was written by **acctwtmp** (entry type=9). Refer to *utmp*(4) in *HP-UX Reference* for more details on this field.
- Exit status for connect session. Refer to *login*(1) and *utmp*(4) in the *HP-UX Reference* for details.
- Number of seconds elapsed from January 1, 1970 to time of entry.
- Date and time of entry.

Fixing wtmp Errors - wtmpfix

Every entry in /etc/wtmp is categorized with a one-digit code in the entry type.

For example, when a user logs into HP-UX, the **login** program stores the value seven (7) in the entry type field of the connect session record. When the same user logs out, an entry type of eight (8) is recorded. You can see this by examining the sample output created by `fwtmp` in the previous section. Note that in the example, login records precede their corresponding logout records in chronological order.

If the system date is changed using *date*(1) while users are logged in, time-stamped ordering can become inconsistent. An inconsistent state can result in logout records to precede login records and can confuse the commands that create connect-session total accounting records.

The command `wtmpfix` fixes corrupted `/etc/wtmp` files by taking `/etc/wtmp` binary records as input, modifying the time/date stamps, and writing the corrected output to a temporary file in binary `wtmp` format. The syntax for `wtmpfix` is:

`wtmpfix` [*files*]

See the *System Administration Tasks* manual and *fwtmp*(1M) in the *HP-UX Reference* for information on using `wtmpfix`. 14

Recording Connect-session Statistics

Total accounting records for connect-session accounting are created by the `acctcon2` command.

Prior to generating the total accounting records, however, you use `acctcon1` to produce columnar output from `/etc/wtmp` and `prctmp` to provide headings for your columnar output. Both `acctcon1` and `acctcon2` are found in *acctcon*(1M) of the *HP-UX Reference*. `Prctmp` is a shell script found in *acctsh*(1M).

Tabulating Output for Connect-session Accounting—acctcon1

Acctcon1 converts the sequence of login and logout records read from /etc/wtmp into output in columnar ASCII format. This output can be supplied as input to prctmp or acctcon2, two commands that generate connect-session reports.

The columnar output of acctcon1 resembles that of fwtmp (shown previously), but sorts by different categories, including device address and duration of connect time.

```
$ acctcon1 < /etc/wtmp
20095488   353   sarah   1665   2478   479518335  Tue Mar 12 16:32:15 1985
521012224  352   mike                  479493590  Tue Mar 12 09:40:00 1985
520095488  351   bill    0      5012   479521475  Tue Mar 12 17:24:35 1985
521011712  0     root    41047  6488   479475173  Tue Mar 12 04:32:53 1985
```

14

The output contains the following information:

- Device address (in decimal equivalent of major/minor device address) at which the connect session occurred.
- User ID for the connect session record.
- User login name.
- Number of prime connect-time seconds used during the connect session.
- Number of non-prime connect seconds.
- Number of seconds elapsed from January 1, 1970 to connect-session starting time.
- Date and starting time time of connect session.

In addition to its default usage, `acctcon1` has four options:

Option	Description
`-p`	This option tells **`acctcon1`** not to produce one record per connect session. Instead, **`acctcon1`** simply echoes its input—one line per **`wtmp`** record—showing line name, login name, and time (in both seconds and day/time format). Using this option is similar to using **`fwtmp`**, except that this option doesn't show status information, whereas **`fwtmp`** does.
`-t`	**`acctcon1`** maintains a list of lines on which users are logged in. When it reaches the end of its input, it emits a session record for each line that still appears to be active. It normally assumes that its input is a current file, so that it uses the current time as the ending time for each session in progress. The **`-t`** flag causes it to use, instead, the last time found in its input, thus assuring reasonable and repeatable numbers for non-current files.
`-l` *file*	This option causes a line usage summary report to be placed in *file*. This report shows each line's name, number of minutes used, percentage of total elapsed time used, number of sessions charged, number of logins, and number of logins and logoffs. This report can be used to keep track of line usage, identify bad lines, and find software/hardware oddities. Note that hang-up, termination of **`login`**, and termination of the login shell each generate logoff records; therefore, the number of logoffs is often three to four times the number of connect sessions.
`-o` *file*	Using the `-o` option (for example, **`acctcon1 -o f_overall`**) creates a concise *file* containing a record of the accounting period, giving starting time, ending time, number of reboots, and number of date changes.

`Acctcon1` options can be combined, as shown in the following example, which uses both `-l` and `-t` options. Also, the standard output of `acctcon1` is redirected to the file `ctmp`:

```
$ acctcon1 -t -l line_use < /etc/wtmp > ctmp
$ cat line_use
TOTAL DURATION IS 899 MINUTES
LINE    MINUTES PERCENT # SESS  # ON    # OFF
console 856     95      1       1       1
tty07   69      8       1       1       1
ttya1   40      4       1       1       1
tty10   84      9       1       1       1
TOTALS  1049    --      4       4       4
```

Heading Columns of acctcon1 Output—prctmp

The `prctmp` command simply puts headings above the output created by `acctcon1`, thus making the report more readable.

14 Since `prctmp` takes its input from standard input, pipe the output from `acctcon1` into `prctmp` as follows:

```
$ acctcon1 < /etc/wtmp | prctmp
```

Creating Connect-session Total Accounting Records—acctcon2

Connect-session total accounting records are created by the `acctcon2` command, found in *acctcon*(1M) of the *HP-UX Reference*.

`Acctcon2` takes standard input created by `acctcon1` and converts it into the structure of `tacct.h`, explained in *acct*(1M) of the *HP-UX Reference*.

The total accounting records created by `acctcon2` are sent to standard output, so to store these records, you must redirect standard output to a file.

In the following command, the total accounting records generated by `/etc/wtmp` is redirected into the file `ctacct`:

```
$ acctcon1 < /etc/wtmp | acctcon2 > ctacct
```

Process Accounting

Process accounting quantifies resource use by process (that is, by programs executed). Statistics include memory usage, CPU time, number of input/output transfers for individual processes. This information is accumulated in a process-accounting file, called by default `/usr/adm/pacct`. Note that several accounting commands do not work properly unless this name is used.

Information contained in `/usr/adm/pacct` includes:

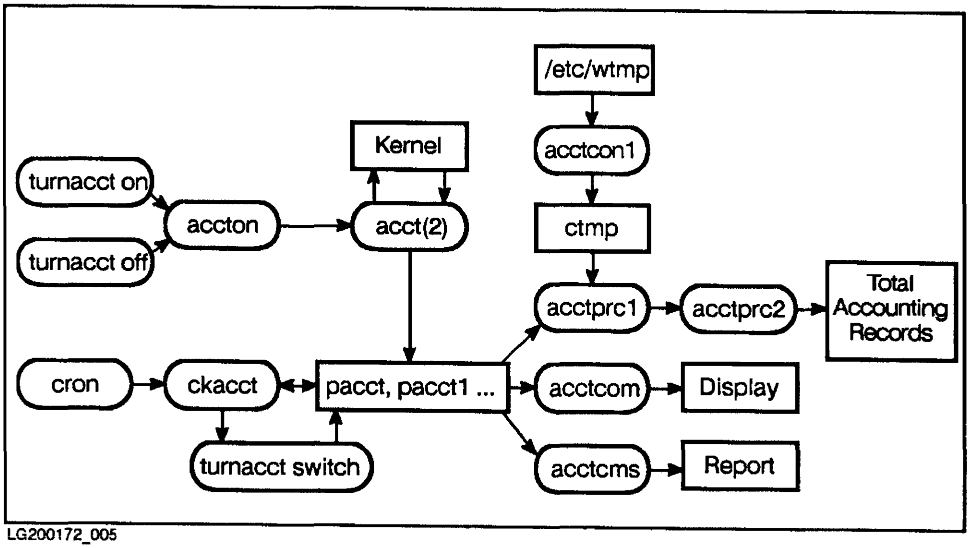
- the user ID of the process's owner
- the name of the command that invoked the process
- the amount of time it took the process to execute

Once system accounting is installed and turned on, the termination of every process is recorded. The kernel writes a record of the terminating process into the current process accounting file, `/usr/adm/pacct`.

For greater detail on the contents and format of process accounting records, refer to *acct*(4) in the *HP-UX Reference*.

Several accounting commands exist to display, report, and summarize process accounting information. In addition, certain commands turn process accounting on or off and ensure that `/usr/adm/pacct` does not grow out of bounds.

The following figure shows the flow of information and commands involved in process accounting:



14

Figure 14-6. Process Accounting

Turning On Process Accounting

Before System Accounting can generate process accounting data, process accounting must be turned on. This is done with the `startup` command, run from the system initialization shell script `/etc/rc`. If you have updated `/etc/rc` for System Accounting (as described earlier in "System Accounting Set-up Guidelines"), process accounting is activated automatically and you seldom need to use the commands described here. These commands are described in case you ever need to manually turn on or off process accounting.

Process accounting can be turned on and off using either of two commands: `turnacct` and `accton`. `Turnacct` is a multi-faceted shell script found in *acctsh*(1M) and `accton` is found in *acct*(1M), both located in the *HP-UX Reference*. (Both commands are used to turn accounting off also, but we will discuss that later.)

`turnacct on` — `Turnacct on` is the command used most often to activate accounting. Only the superuser and the `adm` login can execute this command.

`Turnacct on` works as follows:

1. Checks to see if the process accounting file /usr/adm/pacct exists, and if not, turnacct creates the file.
2. Turns on process accounting by invoking accton with the default filename argument /usr/adm/pacct. This causes all process accounting data to be stored in the pacct file.

accton — Accton, specified with a filename, turns on process accounting, calls *acct*(2), and has the kernel write process-accounting records to the file. Note that the specified file must exist before executing accton; otherwise, accton fails. Only the superuser can execute accton.

If no file name is given, process accounting is turned off; the kernel stops writing process accounting records to the process accounting file.

14

Turning Off Process Accounting

If you updated the /etc/shutdown shell script to include shutacct (as described earlier in "When System Accounting is Turned Off") you seldom use these commands, because the shutacct command automatically turns off process accounting. The following commands are explained, should you ever need to deactivate process accounting manually.

Process accounting can be turned off by using either of two commands: turnacct off and accton (with no filename argument). Both commands tell the kernel to stop writing records to the current process accounting file.

turnacct off — Turnacct off turns off process accounting by invoking the accton command without the optional filename argument. Turnacct off is executed only by the superuser and the adm login.

accton — When accton is invoked without the optional filename argument, process accounting is turned off.

As shown in the System Data Flow Diagram, accton tells the kernel to stop writing process accounting records by using the system call acct.

Managing the Size of the Process Accounting File pacct

On a multi-user system, many processes execute during a single hour, causing process accounting files to become quite large. System Accounting has built-in mechanisms that ensure that the default process accounting file `/usr/adm/pacct` does not grow out of bounds.

The two commands used for this purpose are: `turnacct switch` and `ckpacct`. Note that these commands work only on the default process accounting file, `/usr/adm/pacct`.

Checking the Size of pacct—ckpacct

The command `ckpacct` is typically invoked by `cron` hourly to ensure that the current process accounting file `/usr/adm/pacct` does not become too large. If disk space becomes critically short, process accounting is turned off until sufficient space is available.

14

The syntax of `ckpacct` is:

`ckpacct` [*blocks*]

If the size of `pacct` exceeds the *blocks* argument, 1 000 by default if *blocks* is not specified, then `turnacct switch` is executed. `turnacct switch` renames the current `pacct` file and creates a new `pacct` file.

If the amount of free space falls below a threshold of 500 blocks, `ckpacct` automatically turns off process accounting via `turnacct off`. When free space exceeds the specified threshold, process accounting is reactivated.

The kernel can also enforce a size limit on the size of `pacct`. This takes precedence over the limit set by `ckpacct`. Refer to `acctsh(1M)` and *acct*(2) in the *HP-UX Reference* for more details.

If `/usr/adm/pacct` becomes too large, a new `/usr/adm/pacct` file is created via `turnacct switch`.

The `turnacct switch` renames the current `/usr/adm/pacct` file so that it no longer receives data, and creates a new, empty `/usr/adm/pacct` file.

Changing Process Accounting Files—turnacct switch

`turnacct switch` is used to create a new `/usr/adm/pacct` file when the current `/usr/adm/pacct` file is too large. The action of `turnacct switch` can be summarized as follows:

1. Process accounting is temporarily turned off.
2. The current `pacct` file is renamed to `pacct`*incr*, where *incr* is a number starting at *1* and incrementing by one for each additional `pacct` file that is created via `turnacct switch`.
3. After the old `pacct` file is renamed to `pacct`*incr*, a new, current `pacct` file is created.
4. Process accounting is restarted; the kernel starts writing records to the newly created `pacct` file.

14

Displaying Process Accounting Records - acctcom

The *acctcom*(1M) command allows you to display records from any file containing process accounting records. `Acctcom` generates a columnar report with descriptive headings for each column. Each line of the report represents the execution statistics accumulated by a particular process during its lifetime. Typically you use this command to display records from the `/usr/adm/pacct` files (`pacct`, `pacct1`, `pacct2` ...).

`Acctcom` is a very versatile command with many options. Its syntax follows:

`acctcom` [[*options*] [*file*]] ...

If no *file* is specified, `acctcom` uses the current `/usr/adm/pacct` file as input. Input can also be taken from standard input. Some `acctcom` options allow you to select only the records that you want to see; other options control the format of the report.

Sample Report using acctcom

The following sample report illustrates the default use of acctcom.

```
$ acctcom
ACCOUNTING RECORDS FROM:  Thu Mar 21 12:52:26 1985
COMMAND                      START    END           REAL     CPU    MEAN
NAME       USER     TTYNAME  TIME     TIME          (SECS)  (SECS) SIZE(K)
#accton    root     console  12:52:26 12:52:26      0.12     0.10   19.00
ls         sarah    tty07    14:04:08 14:04:08      0.28     0.23   16.50
ckpacct    adm      ?        14:30:00 14:30:05      5.13     1.45   24.00
pwd        bill     tty10    15:09:07 15:09:07      0.48     0.22   22.50
find       sarah    tty07    18:51:37 18:51:39      2.73     0.15   26.50
tabs       root     console  19:10:18 19:10:18      0.92     0.13   23.50
stty       root     console  19:10:19 19:10:19      0.88     0.08   26.00
mail       bill     tty10    19:10:21 19:10:22      1.78     0.23   28.50
news       root     console  19:10:23 19:10:23      0.73     0.12   23.00
acctcom    adm      ttya0    19:53:16 19:53:38     22.58     2.55   28.50
#date      root     console  19:58:26 12:58:26      0.12     0.10   19.00
```

14

Definitions of Information Produced by acctcom

Default output of acctcom' includes the following:

Column Header	Definition
`COMMAND NAME`	The name of the command or program that spawned the process. Commands *requiring* superuser privileges are listed with a preceding # (see **`#date`** in sample output above). This feature ensures that actions that can only be accomplished as superuser are visible via system accounting records.
`USER`	The login name of the user who created the process.
`TTYNAME`	The terminal from which the process was executed. If the process was not executed from a known terminal (for example, if it was executed via **`cron`**), a question mark(*?*) appears.
`START TIME`	When the process began executing, specified in *hh:mm:ss* format.
`END TIME`	When the process finished executing.
`REAL (SECS)`	How many seconds elapsed from `START TIME` to `END TIME`.
`CPU (SECS)`	How much CPU time a process used while executing.
`MEAN SIZE(K)`	How much memory (estimated in kilobytes) a process used while executing.

The `MEAN SIZE` estimate is determined from the current process's memory usage at each system clock interrupt, and is therefore subject to statistical sampling errors. Only the memory resident pages of a process are counted; no pages in the swap space are counted. Shared code and data are divided among the processes using them. MEAN SIZE is divided by the number of processes sharing the code or data.

Additional Capabilities of acctcom

Listed below are columns not displayed in a standard `acctcom` report, but can be displayed by using `acctcom` options. The `acctcom` options used to generate these columns are described in the next section.

Column Header	Definition
F	For a process created by **fork** that does not call **exec**, this column takes the value *1*; otherwise, this column shows a *0*.
STAT	This column displays the system exit status if the process terminated by calling **exit**. When a process terminates normally, this field shows a *0*. If a command terminates abnormally, the value returned is the signal number that caused the process to terminate. If a core file image is produced by the signal, the value is a signal number plus 0200. For specific details of signal values, see *signal*(5) in the *HP-UX Reference*.
HOG FACTOR	The hog factor is computed as the CPU time divided by REAL time; it provides a relative measure of the available CPU time used by the process during its execution. For example, a hog factor of less than 0.50 indicates that the process spent less than half of its time using the CPU. A hog factor of 0.75 indicates that a process spent 75% of its time using the CPU.
KCORE MIN	This calculation provides a combined measurement of the amount of memory used (in kilobytes) and the length of time it was used (in minutes). It is computed as follows: `KCORE MIN = CPU (SECS) * MEAN SIZE(K) / 60`
CPU SYS	This is the portion of total CPU time that was spent executing operating system code, such as system calls (for example, writing to disk).
USER (SECS)	This is the remaining portion of CPU time. User CPU time is the amount of time actually spent executing a process's code (rather than system code).

Column Header	Definition
CPU FACTOR	Whenever you execute a command, the CPU spends part of its time actually executing the command's code (user CPU time) and spends the rest of its time performing system functions, such as writing to the disk or terminal (system CPU time). That is, total CPU time is comprised of both CPU SYS and USER CPU time: `CPU (SECS) = CPU SYS + USER (SECS)` The CPU factor shows the ratio of user CPU time to total CPU time: `CPU FACTOR = USER (SECS) / (CPU SYS + USER (SECS))` For example, if a command has a CPU factor of 0.35, that means that the CPU spent 35% of its time executing user code and 65% performing system functions.
CHARS TRNSFD	The number of characters (bytes) read and/or written by the command is displayed in this column.
BLOCKS R/W	This column shows the number of file system blocks read and/or written as a result of executing this command. *This number is not directly related to* CHARS TRNSFD *and may vary each time the command is executed*, because BLOCKS R/W is affected by directory searches made before opening files, other processes accessing the same files, and general file system activity.

Formating Options of acctcom

The options to *acctcom*(1M) enable additional statistics to be reported, some of which are discussed in the previous section:

Option	Description
`-a`	Display at the end of the report statistical averages. Total number of commands processed shown as (`cmds=xxx`) ■ average real time per process (`Real=x.xx`) ■ average CPU time per process (`CPU=x.xx`) ■ average USER CPU time per process (`USER=x.xx`) ■ average SYS CPU time per process (`SYS=x.xx`) ■ average characters transferred (`CHARS=x.xx`) ■ average blocks transferred (`BLK=x.xx`) ■ average CPU factor (`USR/TOT=x.xx`) ■ average HOG factor (`HOG=x.xx`)
`-b`	Display the process records in reverse order: most recently executed commands shown first.
`-f`	Print the `fork/exec` flag (`F` column) and process exit status (`STAT` column).
`-h`	Display the optional `HOG FACTOR` column, instead of the standard mean memory size `MEAN SIZE(K)`.
`-i`	Replace the standard `MEAN SIZE(K)` column with the optional I/O counts—`CHARS TRNSFD` and `BLOCKS R/W`.
`-k`	Replace the standard `MEAN SIZE(K)` column with `KCORE MIN`.
`-m`	Show the default column `MEAN SIZE(K)` when `MEAN SIZE(K)` is superceded by another option, such as `KCORE MIN`.
`-r`	Include the optional `CPU FACTOR` column in the report.
`-t`	Show separate CPU times for system (`CPU SYS`) and user (`USER (SECS)`).
`-v`	Suppress printing column headings.
`-q`	This option produces only the averages as displayed by the `-a` option, without the individual process accounting records.
`-o` *ofile*	Copy the input process accounting records to *ofile*.

Record Selection Options of acctcom

The options described here allow you to select the records that are included in the report produced by `acctcom`. See your *System Administration Tasks* manual for usage examples.

Option	Description
`-l` *line*	Display only the processes executed from the user terminal `/dev/`*line*.
`-u` *user*	Show only the processes belonging to *user*.
`-g` *group*	Show only processes belonging to *group*, which may be specified by either group name or group ID.
`-s` *time*	Select processes existing *at or after time*. Time is specified in 24-hour format—*hr*[:*min*[:*sec*]].
`-e` *time*	Select processes existing *at or before time*. Time is specified in 24-hour format *hr*[:*min*[:*sec*]].
`-S` *time*	Select processes *starting* at or after time where *time* is in 24-hour format.
`-E` *time*	Display only processes *terminated* at or before *time*, where time is in 24-hour format *hr*:[*min*[:*sec*]]. Note both the `-S` and `-E` options with the same *time* argument cause `acctcom` to display only processes existing at the specified *time*.
`-n` *pattern*	Show only the commands matching *pattern*. *pattern* can be a regular expression as described in *ed*(1), except that + means one or more occurrences.
`-H` *factor*	Display only processes whose hog factor exceeds *factor*.
`-O` *time*	Show only processes whose system CPU time exceeds *time*, specified in seconds.
`-C` *sec*	Show only processes whose total CPU time (`SYS + USER`) exceeds *sec* seconds.
`-I` *chars*	Display only processes transferring more characters than the limit given by *chars*.

Process-accounting Report of Commands - acctcms

The *acctcms*(1M) command takes `/usr/adm/pacct` as input, and produces summary accounting information by command instead of process. The action of *acctcms*(1M) can be summarized as follows:

1. `acctcms` reads the data in process-accounting records to accumulate execution statistics for each unique command name. This information is stored in internal summary format—one record per command name.
2. Command summary records created in step 1 are sorted according to the `acctcms` options specified.
3. Command summary records are written to standard output in the internal summary format mentioned in step 1. This format is not readable.

14

The syntax of the `acctcms` command is:

`acctcms` [*options*]`files`

where *files* is a list of the input process accounting files for which the command summary report is to be generated. The *options* are discussed in subsequent sections.

Producing a Readable Report - the -a option

By default, the output of `acctcms` is in internal summary record format; if you display it to your terminal, all you see is gibberish. To get a ASCII, readable report, use the `-a` option.

The `-a` option causes `acctcms` to produce a report with descriptive column headings. Total and average (mean) execution statistics for each command are displayed—one line per command—along with total and average statistics over all commands in the report.

Sample Report by acctcms

The ASCII reports produced by `acctcms` contain more than 80 characters per line. When displayed at an 80-column terminal, the lines wrap around on the screen, and if printed on an 80-column printer, some of the rightmost columns of the report are lost. Therefore, compensate as follows:

- Use a printer with compressed print capabilities, so that the entire report fits on standard computer paper; or
- Use a wide-capability printer to display all the information, such as one that prints 132 columns.

The following example shows a command summary report for the current process accounting file (if no file is specified, the `/usr/adm/pacct` file is used). Note also that total execution statistics for all commands are grouped under the command name `TOTALS`.

```
$ acctcms -a proc.figs
                              TOTAL COMMAND SUMMARY
COMMAND NUMBER    TOTAL     TOTAL      TOTAL  MEAN      MEAN     HOG    CHARS BLOCKS
NAME      CMDS KCOREMIN   CPU-MIN REAL-MIN SIZE-K   CPU-MIN   FACTOR TRNSFD   READ

TOTALS      61    17.63   0.38      164.49  46.25     0.01     0.00 104553   1027

acctcms     17    12.13   0.16        0.35  76.72     0.01     0.45  49192    306
sh           8     2.43   0.09      152.86  26.79     0.01     0.00   9043    163
more         3     0.73   0.02       10.50  31.00     0.01     0.00  21618     83
ll           6     0.61   0.04        0.11  16.50     0.01     0.33   5715     95
acctcom      4     0.58   0.02        0.07  28.50     0.01     0.30  15319     42
cat          4     0.19   0.01        0.35  22.97     0.00     0.02   3112     52
rm           2     0.11   0.00        0.02  22.22     0.00     0.29      0     29
chmod        2     0.10   0.00        0.01  22.00     0.00     0.35      0     15
accton       2     0.08   0.00        0.02  19.00     0.00     0.29      0     22
sed          2     0.08   0.01        0.04  14.50     0.00     0.13     73     38
echo         2     0.05   0.00        0.02  20.00     0.00     0.16     22     21
```

Header Descriptions

Descriptions of the columnar data produced by `acctcms` are shown in the following table.

Column Header	Description
COMMAND NAME	The name of the command for which execution statistics are summarized. Unfortunately, all shell procedures are lumped together under the name sh, because only object modules are reported by the process accounting system.
NUMBER CMDS	The total number of times that the command was invoked.
TOTAL KCOREMIN	The total amount of kcore minutes accumulated for the command. (Refer to the section "Additional acctcom Capabilities" earlier in this chapter for more information.)
TOTAL CPU-MIN	The total CPU time allocated to the named command.
TOTAL REAL-MIN	Total allocated real-time minutes are displayed in this column.
MEAN SIZE-K	The average amount of memory (in kilobytes) consumed by the command.
MEAN CPU-MIN	The average CPU time allocated per command invocation is shown here; the following equation shows how it is computed: `MEAN CPU-MIN = TOTAL CPU-MIN / NUMBER CMDS`
HOG FACTOR	The average hog factor over all invocations of the command. It is computed as: `HOG FACTOR = TOTAL CPU-MIN / TOTAL REAL-MIN`
CHARS TRNSFD	The total number of characters transferred by the command. Note that this number may sometimes be negative.
BLOCKS READ	A total count of the physical blocks read and written by the given command. (Refer to the section "Displaying Process Accounting Records—acctcom" in this chapter for details on the significance of this total.)

Note

When only the -a option is specified, the report is sorted in descending order on the TOTAL KCOREMIN column: commands using more TOTAL KCOREMIN are shown before those using fewer TOTAL KCOREMIN. This gives a relative measure of memory used over time by the various commands: commands listed early in the report use more memory resources than commands listed later.

Additional acctcms Options

Several options besides `-a` can be used to control the format of reports generated by `acctcms`. Some options specify on which field to sort the report; other options control the printing of prime/non-prime time usage.

Option	Description
`-c`	Sort the commands in descending order on `TOTAL CPU-MIN`, rather than by default `TOTAL KCOREMIN`. This report can be used to determine which commands are using most of the computer's CPU time.
`-n`	Cause the report to be sorted in descending order on the column named `NUMBER CMDS`. Commands toward the start of this report are the ones used most frequently; commands toward the end are used least often.
`-j`	Combine all commands invoked only once on one line of the report. This line is denoted by having "`***other`" in the `COMMAND NAME` column. This option is useful for shortening a report that has many one-invocation commands.
`-o`	*Used only with the* `-a` *option*, `-o` causes the report to be generated only for commands that were executed during non-prime time (as specified in the `holidays` file). You can use this option to get a non-prime time command summary report.
`-p`	*Also used only with the* `-a` *option*, `-p` elicits a report generated only for commands that were executed during prime time (as specified in `holidays`). This option is used to get a prime time command summary report.

14

Option	Description
-apo	When the options -o and -p are used together with -a, a combination prime and non-prime time report is produced. The output of this report is identical to that produced by -a alone, except that the NUMBER CMDS, TOTAL CPU-MIN, and TOTAL REAL-MIN columns are divided into two columns—one for prime time totals, the other for non-prime time. (Prime time columns have a (P) header, while non-prime time columns are headed by (NP).)
-s [*files*]	Specifies that any named input *files* following the -s on the command line are already in internal summary format. This option is useful for merging previous acctcms reports with current reports. The following example uses -s to create a command summary report from previous process accounting files (pacct?) and the current process accounting file (pacct). The final ASCII report is stored in the file ascii_summary.

```
$ acctcms pacct? > old_summary
$ acctcms pacct > new_summary
$ acctcms -as old_summary new_summary > ascii_summary
```


Creating Total Process-accounting Records

Two commands—acctprc1 and acctprc2—are used to create total accounting records from the process accounting files. The output from acctprc1 is supplied as input to acctprc2, which produces the total accounting records. These commands are invoked by runacct to produce daily accounting information.

acctprc1

Acctprc1 produces readable process-accounting information, mainly for input into acctprc2. Acctprc1 reads process accounting records from standard input, adds login names corresponding to the user ID of each record, and for each process writes an ASCII line showing:

- the user ID of the creator of the process
- the user's login name
- prime CPU time in ticks (a "tick" is one fiftieth of a second)
- non-prime CPU time, also in ticks
- mean memory size (in pages—typically 4 Kbytes)

The format of acctprc1 is:

acctprc1 [*ctmp*]

where ctmp contains a list of login sessions of the form created by acctcon1, sorted by user ID and login name.

Note The number of sessions should be 1000 or less. Running more than 1000 sessions causes the accounting system to "hang" (that is, suspend indefinitely). If that happens, sessions must be killed using kill command and restarted.

To use acctprc1, input must be redirected from a process accounting file, such as /usr/adm/pacct. The following example creates a file, ascii_ptacct, containing ASCII process accounting information useful for creating process total accounting records.

```
$ acctprc1 <pacct >ascii_ptacct
```

Normally, acctprc1 gets login names from the password file /etc/passwd, which is sufficient on systems where each user has a unique user ID. However, on systems where different users share the same user ID, the ctmp file should be specified; it helps acctprc1 distinguish different login names that share the same user ID.

acctprc2

Acctprc2 takes input produced by acctprc1, summarizes the records by user ID and name, and writes the sorted summaries to standard output as total accounting records. The following example creates total accounting records for all processes in the current process accounting file pacct; the total accounting records are stored in the file ptacct.

```
$ acctprc1 <pacct |acctprc2 >ptacct
```

User Fees – chargefee

Although you can use system accounting reports as input to other fiscal accounting packages, HP-UX system accounting also provides some capability for charging fees. `Chargefee`, a shell script on *acctsh*(1M) of the *HP-UX Reference*, allows you to charge fees to specific users by allowing you to tally generic *units* per specific login name.

For example, you might apportion fees among several departments for use of computer resources or charge fees to users for fixing damaged files.

The syntax of this command is:

`chargefee` *login_name number*

where *login_name* represents the user to whom *number* units are being charged and *number* represents how many units to charge to the user.

14

Note *number* can be any whole number from -32K to 32K; when charging fees, remember that a user's total fees must be within this range.

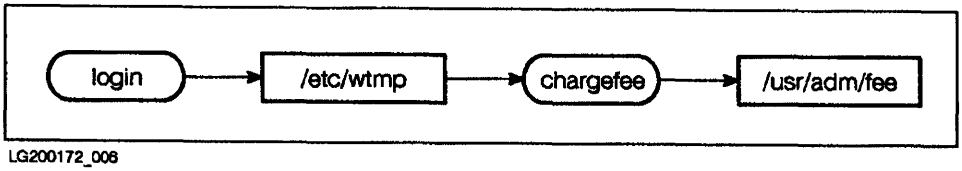


Figure 14-7. Charging Fees

`Chargefee` accumulates fee charge records in the file `/usr/adm/fee`. These records are then merged with other accounting records via `runacct`.

See the *System Administration Tasks* manual for detailed implementation.

Accounting Summaries and Reports

Five commands and shell scripts summarize and report data for a variety of accounting purposes:

`prtacct`	Display total accounting records.
`acctmerg`	Combine total accounting files.
`runacct`	Generate daily summary files and reports.
`prdaily`	Create and display daily summary files and reports generated from `runacct` output.
`monacct`	Create monthly summary files and reports generated from `runacct` output.

In terms of importance, you will probably find that `runacct` is your main daily accounting tool. For explanatory purposes, however, we first discuss `prtacct` and `acctmerg`.

14

Displaying Total Accounting Records - prtacct

The `prtacct` command allows you to display the contents of a process accounting file by taking as input total accounting records and displaying them in ASCII-readable format. Its syntax is

`prtacct` *file* "*heading*"

where *file* is the name of the total accounting file being displayed and "*heading*" is a descriptive header produced by `prtacct`. Since "*heading*" can be a phrase, enclose it in double quotes.

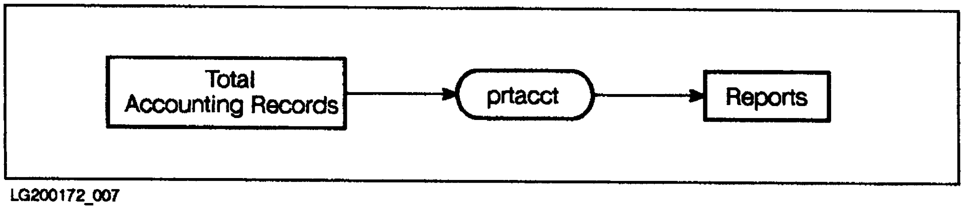


Figure 14-8. Displaying Total Accounting Records

Sample Output by prtacct

Although `prtacct` is capable of providing merged summary reports, this first example displays disk total accounting records only. First, the total accounting records are created by executing disk space accounting commands; then, they are displayed using `prtacct`. When studying this report, note the following:

- There are many similarities between this and the sample report produced by `diskusg` (refer to the section "Disk Space Usage Accounting" in this chapter).
- Since the total accounting records were created only from disk space usage accounting commands, only columns relating to disk space usage have non-zero values.

14

```
$ for file_system in `cat /etc/checklist`
> do
> diskusg $file_system >dtmp.`basename $file_system`
> done
$ diskusg -s dtmp.* |sort +0n +1 |acctdisk >disktacct
$ prtacct disktacct "DISK TOTAL ACCOUNTING RECORDS"

Mar 26 17:01 1985  DISK TOTAL ACCOUNTING RECORDS Page 1

    LOGIN   CPU (MINS)    KCORE-MINS   CONNECT (MINS) DISK   # OF  # OF # DISK FEE
UID NAME   PRIME NPRIME PRIME NPRIME PRIME    NPRIME BLOCKS PROCS SESS SAMPLES
0   TOTAL  0     0      0     0      0        0      11598  0     0    10       0
0   root   0     0      0     0      0        0      10616  0     0    1        0
1   bin    0     0      0     0      0        0      778    0     0    1        0
4   adm    0     0      0     0      0        0      96     0     0    1        0
350 fred   0     0      0     0      0        0      14     0     0    1        0
351 bill   0     0      0     0      0        0      32     0     0    1        0
352 mike   0     0      0     0      0        0      20     0     0    1        0
353 sarah  0     0      0     0      0        0      16     0     0    1        0
354 molly  0     0      0     0      0        0      22     0     0    1        0
355 julie  0     0      0     0      0        0      2      0     0    1        0
501 guest  0     0      0     0      0        0      2      0     0    1        0
```

Report Format of prtacct

`Prtacct` produces a columnar report with one line per total accounting record, with the following column headings:

Column Header	Description
UID	User ID of the user for whom the total accounting record was created.
LOGIN NAME	The user's login name.
CPU (MINS)	Total amount of CPU time (in minutes) consumed by the user, during prime and non-prime time.
KCORE-MINS	This represents a cumulative measure of memory and CPU time consumed by a user during prime and non-prime time. (See also "Additional acctcom Capabilities" earlier in this chapter for a more information).
CONNECT (MINS)	The amount of time a user spent logged into the system, cited by PRIME and NPRIME time.
DISK BLOCKS	The total number of disk blocks allocated to the user.
# OF PROCS	The total number of processes spawned by the user.
# OF SESS	How many times the user logged in.
# DISK SAMPLES	How many times disk accounting was run to obtain the average number of disk blocks listed in the DISK BLOCKS column.
FEE	The number of fee units charged via **chargefee**.

Origin of Columnar Data by prtacct

Process-accounting commands are used to tabulate data in the CPU (MINS), KCORE-MINS, # OF PROCS columns.

Connect-session accounting commands are used to tabulate data in the CONNECT (MINS) and # OF SESS columns.

Disk-space accounting commands are used to tabulate data in the DISK BLOCKS and # DISK SAMPLES columns.

Merging Total Accounting Files - acctmerg

Acctmerg combines the contents of separate disk, process, and connect session total accounting files into a single comprehensive total accounting file. Data is consolidated by user name and ID.

Acctmerg is typically executed by **runacct**.

`Acctmerg` reads standard input and up to nine additional files, all in total accounting record format. Thus, `acctmerg` accepts input from:

- `acctdisk` for disk-usage accounting
- `acctcon2` for connect-session accounting
- `acctprc2` for process accounting.

The syntax for `acctmerg` is:

`acctmerg` [*options*] [*file*] ...

where *options* control the report format and how records are merged and *file* can be one to nine files (plus standard input) being merged into a single total accounting file, written to standard output.

Command Options for acctmerg

14

The following options can be used to control the report format and how the total accounting records are combined. (For more details, see the *acctmerg*(1M) manual page in *HP-UX Reference*.)

`-a` Produce output in ASCII, much like that of `prtacct` reportage, but without report headings.

`-i` Accept input to `acctmerg` in ASCII format.

`-p` Echo input records without merging or processing, and display output as with the `-a` option.

`-t` Produce a single total accounting record of all input. For the ASCII version of this record, use the both `-t` and `-a` options, each specified with its own hyphen.

```
$ acctmerg -t -a <tot_acct_recs
```

`-u` Merge records by user ID only, rather than user ID and name.

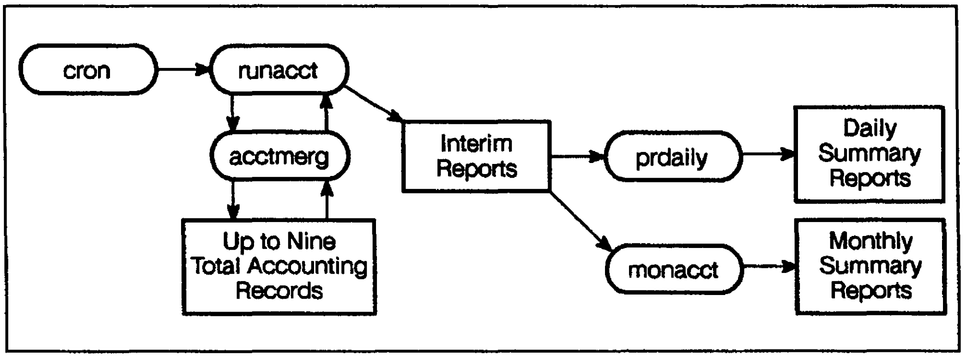
`-v` Produce verbose ASCII output, with more precise notation (showing full logarithmic and exponential factors) for floating point numbers.

Creating Daily Accounting Information - runacct

Runacct is a comprehensive daily accounting procedure, a shell script found on the *acctsh*(1M) manual page of the *HP-UX Reference*. Typically, runacct is invoked daily by cron during non-prime hours when users are logged off. This ensures for maximum accuracy.

As shown in the following diagram, runacct processes and merges disk, connect session, process, and fee accounting data, and produces summary files and reports. It prepares cumulative summary files for use by prdaily and for billing purposes.

14



LG200172_008

Figure 14-9. How runacct Generates Summary Reports

This section discusses the following aspects of runacct:

- files processed by runacct
- the states through which runacct progresses while executing
- recovery from runacct failure
- restarting runacct
- reports produced by runacct

Files Processed by runacct

The following files, processed by `runacct`, are of particular interest to the reader. (File names are given relative to the directory `/usr/adm/acct`.)

`nite/lineuse` Usage statistics for each terminal line on the system. This report is useful for detecting bad lines. If the ratio of logoffs to logins on a particular line exceeds 3 to 1, that line is likely failing.

`nite/daytacct` Total accounting records from the previous day.

`sum/tacct` Accumulated total accounting records for each day's total accounting records (`nite/daytacct`). `sum/tacct` can be used for billing purposes. It is restarted each month or fiscal period by the `monacct` shell script.

14 `sum/daycms` Produced by `acctcms`, `sum/daycms` contains the daily process-accounting command summary. The ASCII version of this file is in `nite/daycms`.

`sum/cms` Accumulates each day's command summaries (`sum/daycms`). A new `sum/cms` file is created each month by `monacct`. The ASCII version of this file is in `nite/cms`.

`sum/loginlog` Maintains a record of the last time each user logged in.

`sum/rprt`*mmdd* Main daily accounting report created by `runacct`. The name for this report is created automatically by the system with *mm* being the month and *dd* the day of the report. This report can be printed via `prdaily`.

Safeguards of runacct

Runacct does not damage files. A series of protective mechanisms recognize errors, provide intelligent diagnostics, and terminate processing so that runacct can be restarted with minimal intervention. To ensure these goals, the following actions are performed by runacct:

- Runacct's progress is recorded by writing descriptive messages to the nite/active file.
- All diagnostic output generated while runacct executes is redirected to the file nite/fd2log.
- If the files lock and lock1 exist when runacct is invoked, an error message is displayed and execution terminates.
- The lastdate file contains the month and day when runacct was last run and is used to prevent runacct from being executed more than once per day.
- If runacct detects an error, a message is written to /dev/console, mail is sent to root and adm, locks are removed, diagnostics files are saved, and execution is terminated.

States of runacct

Runacct accomplishes its tasks by proceeding through various states, each associated with specific tasks. These states enable runacct to be restarted easily.

For example, in one state, total accounting records for connect sessions are created; in another, disk, connect session, process, and fee total accounting records are merged to create one total accounting file.

As runacct executes, it records its progress by writing the name of the most recently completed state into the /usr/adm/statefile file. After processing for each state ends, runacct examines statefile to determine which state to enter next. When runacct reaches the final state (CLEANUP), the lock and lock1 files are removed, and execution terminates.

State	Action
SETUP	Execute the command **turnacct switch**. The process accounting files, **pacct?**, are moved to Spacct?.*mmdd*. The **/etc/wtmp** file is moved to **nite/wtmp**.*mmdd* with the current time added on the end.
WTMPFIX	Check **nite/wtmp**.*mmdd* for correctness by **wtmpfix**. Since some some date changes cause **acctcon1** to fail, **wtmpfix** adjusts time stamps in **nite/wtmp**.*mmdd* when it encounters a date-change record.
CONNECT1	Write connect-session records to **ctmp**. The **lineuse** file is created, and the **reboots** file, showing all boot records found in **nite/wtmp**.*mmdd*, is created.
CONNECT2	Convert **ctmp** to connect-session total accounting records in the file **ctacct**.*mmdd*.
PROCESS	Execute **acctprc1** and **acctprc2** programs to convert the process-accounting files Spacct?.*mmdd* to total accounting records in **ptacct?**.*mmdd*. The **Spacct** and **ptacct** files are correlated by number so that if **runacct** fails, **Spacct** files will not have to be reprocessed. One precaution should be noted: when restarting **runacct** in this state, remove the last **ptacct** file. If you don't, **runacct** will not terminate.
MERGE	Merge process and connect-session total accounting records to form **nite/daytacct**.
FEES	Merge any ASCII **tacct** records from the file **fee** into **nite/daytacct**.
DISK	On the day after the **dodisk** shell script runs, merge **nite/disktacct** with **nite/daytacct**.
MERGETACCT	Merge **nite/daytacct** with **sum/tacct**, the cumulative total accounting file. Each day, **nite/daytacct** is saved in **sum/tacct***mmdd*, so that **sum/tacct** can be recreated if it becomes corrupted or lost.
CMS	Merge today's command summary into the cumulative summary file **sum/cms**. Produce ASCII and internal-format command summary files.

14

State	Action
`USEREXIT`	Execute any installation-dependent (local) accounting programs. For example, you might want to execute commands that generate daily billing data for individual users, such as the shell script `acct_bill` described in *System Administration Tasks* section "Sample System Accounting Shell Scripts".
`CLEANUP`	Clean up the temporary files, run `prdaily` and save its output in the file `sum/rprt`*mmdd*, remove the locks, then exit.

Recovering from runacct Failure

Sometimes `runacct` fails and terminates abnormally. The primary reasons for `runacct` failure are:

- a system crash
- not enough disk space remaining in `/usr`
- a corrupted `wtmp` file

14

For information on how to recover from `runacct` failure, consult the *System Administration Tasks* manual.

Daily and Monthly Reports of runacct

Each time `runacct` is executed, it generates five reports:

- Daily Line Usage Report
- Daily Resource Usage Report
- Daily Command Summary
- Monthly Total Command Summary
- Last Login

Daily Line Usage Report of runacct

The **Daily Line Usage Report** summarizes connect-session accounting since the last invocation of `runacct`. It logs system reboots, power failure recoveries, and any other records dumped into `/etc/wtmp` via `acctwtmp`, and provides an analysis of line utilization.

The `FROM/TO` banner in the first part of the report identifies the period covered. The times are the date-time the last report was generated by `runacct`, and the date-time the current report is generated. It is followed by a log of system reboots, shutdowns, power failure recoveries, and any other records dumped into `wtmp` by the `acctwtmp` command.

The second part of the Daily Line Usage Report is an analysis of line utilization. The `TOTAL DURATION` shows how long the system was in a multi-user state. The columns of the report are defined as follows:

Column	Description
`LINE`	The terminal line or access port being reported on.
`MINUTES`	The total number of minutes that the line was in use during the accounting period.
`PERCENT`	The percentage of `TOTAL DURATION` that the line was in use: `PERCENT = (MINUTES / TOTAL DURATION) * 100`
`# SESS`	The number of times that this port was accessed for a `login` session.
`# ON`	Historically, this column displayed the number of times a port was used to log a user on. However, since `login` is no longer be executed to log in a new user, this column should be identical to `# SESS`.
`# OFF`	This column reflects the number of times a user logs off and any interrupts that occur on the line. Interrupts occur on a port when `getty` is first invoked, which happens when the system is brought to run-level 2. If `# OFF` exceeds `# ON` by a large factor, the multiplexer, modem, or cable is probably deteriorating, or a bad connection exists somewhere. The most common cause is an unconnected cable dangling from the multiplexer.

The `/etc/wtmp` should be monitored regularly, as this file from the source of connect-session accounting data. If `/etc/wtmp` grows rapidly, execute `acctcon1` to determine the noisiest line. If interruptions occur at a high rate, general system performance is affected.

Daily Resource Usage Report

The **Daily Resource Usage Report** summarizes resource usage, by merging all total accounting records and displaying the information by individual user.

This report gives a by-user breakdown of system-resource usage. The format of this report is identical to that produced by the `prtacct` command. (For definitions of the data and format of this report, refer to the discussion of `prtacct` in the "Displaying Total Account Records – prtacct" section of this chapter.)

Daily and Monthly Command Summary

The **Daily Command Summary** shows resource-usage data about individual commands since the last invocation of `runacct`. It is useful in identifying the most widely used commands, and applying that knowledge of command characteristics when tuning your system.

This report is sorted by `TOTAL KCOREMIN`, an arbitrary but useful yardstick for judging CPU usage.

The **Monthly Total Command Summary** is identical to the Daily Command Summary, except for the duration of time reported. The Monthly Total Command Summary summarizes commands from the start of the fiscal period to the current date, and reflects the data accumulated since the last invocation of `monacct`.

The output of these reports is identical to that produced by `acctcms`. For definitions of the data found in this report, refer to the discussion of `acctcms` in the "Process Accounting" section of this chapter.

Last Login

The **Last Login** simply shows the last date and time each user logged into the system. It is useful for finding likely archive material or for removing unused user directories.

Creating Daily Summary Reports - prdaily

`Prdaily` is invoked by `runacct` as `runacct` finishes executing. `Prdaily` formats the previous day's accounting data, and places the report in the file `/user/adm/acct/sum/rpt`*mmdd* where *mmdd* is the month and day of the report.

`Prdaily` can also be used to display any daily report file created by `runacct`.

The syntax of `prdaily` is:

`prdaily` [`-l`] [`-c`] [*mmdd*]

where:

14

- The `-l` option prints a report of exceptional usage by login name for the specified date. This option might reveal which users are consuming excessive amounts of system resources. The limits for exceptional usage are kept in the file `/usr/lib/acct/ptelus.awk` and can be edited to reflect your installation's requirements.
- The `-c` option is valid only for the current day's accounting, and is used to identify exceptional resource usage by command. The limits for exceptional usage are maintained in the file `/usr/lib/acct/ptecms.awk` and can be edited to reflect your system's needs.
- *mmdd* is an optional report date. If no date is specified, `prdaily` produces a report of the current day's accounting information.

Creating Monthly Accounting Reports - monacct

The `monacct` shell script is invoked once a month (or accounting period) to summarize daily accounting files and produce a summary files for the accounting period. The resulting output is stored in the directory `/usr/adm/acct/fiscal`. After creating its monthly reports, `monacct` removes the old daily accounting files from the directory `/usr/adm/acct/sum` and replaces them with new summary accounting files.

`Monacct` is found on the *acctsh*(1M) manual page of *HP-UX Reference*.

The syntax of `monacct` is

`monacct` *number*

where *number* specifies month or period (01=January, 12=December). If *number* is not specified, `monacct` assumes it is being invoked for the current month. This default is useful if `monacct` is executed via `cron` on the first day of each month.

14

Descriptions of the files created in the `acct/fiscal` directory follow:

- `cms`*?* contains the total command summary file of the accounting period denoted by *?*. The file is stored in internal summary format. To display this file, use the `acctcms` command. The following example shows how to display this file for June:

  ```
  $ acctcms -a -s /usr/adm/acct/nite/fiscal/cms06
  ```

- `fiscrpt`*?* contains a report similar to that produced by `prdaily`. The report shows line and resource usage for the month represented by *?*. To display the fiscal accounting file for November, enter:

  ```
  $ cat /usr/adm/acct/nite/fiscal/fiscrpt11
  ```

- `tacct`*?* is the total accounting file for the month represented by *?*. To display this file, use the `prtacct` command. The following example shows how to display the total accounting summary file for January:

  ```
  $ prtacct /usr/adm/acct/fiscal/tacct01 "JANUARY TOTAL ACCOUNTING"
  ```

Disk Quotas

Disk quotas are used to limit the number of files and file blocks a user can own per file system. Using disk quotas, the system administrator can assign file and block quotas to users on any writable file system.

HP-UX disk quotas work on both standard and NFS-mounted file systems and in a clustered environment (Series 300/400/700 only).

Disk quotas consist of tools to

- manage disk resources equitably by limiting disk usage
- help prevent users from accidentally filling a file system

Planning to Set up Disk Quotas

14 Disk quotas are already configured into your operating system. Using disk quotas involves two basic steps:

- determining which file systems warrant disk quotas
- establishing soft limits (quotas) and hard limits (limits) on users' disk consumption

Use the `quotas` option to `/etc/checklist` entries and the *quotacheck*(1M) command to `/etc/bcheckrc` to have the system invoke disk quotas automatically at boot time.

Deciding Which File Systems Require Disk Quotas

You need to decide which file systems warrant disk-quota controls. Typically, only file systems containing users' home directories and possibly the `/usr` file system require disk quotas. Do not assign quotas to the `/tmp` directory, which is historically a less restricted environment that is periodically purged of excess files by other means.

You can set disk quotas on all users on a file system, or only specific users, by using the `edquota` command, discussed later in this chapter. You can also establish separate limits for the number of files (designated as inodes) and number of blocks (given in units of one kilobyte) per user.

When setting up disk quotas, be sure these file systems have adequate available space for data collection. Note that the `quotas` file grows based on the numeric value of the user ID. Therefore, if you are setting up disk quotas, you might want to keep the numeric value of your user IDs to low numbers.

Setting Quotas and Limits

Imposing disk quotas involves setting soft and hard limits on the amount of file space a user can consume. The **soft limit**, or **quota**, is the amount of file space allocated for a given user, and can be exceeded temporarily. The **hard limit**, or **limit**, is the maximum amount of file space allowed by a given user.

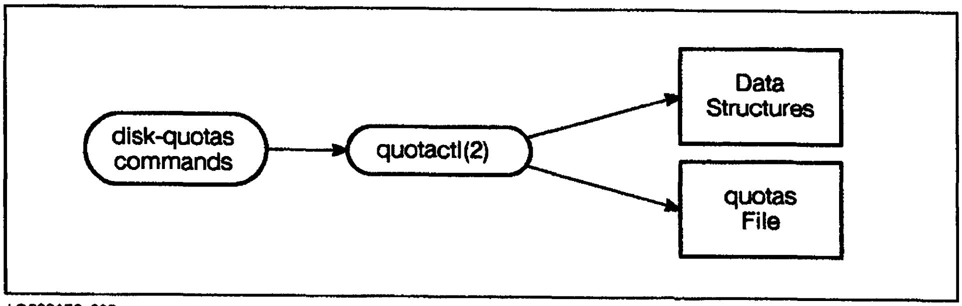
The grace period for exceeding the quota is set by the system administrator, or left at the default of one week. Once the grace period expires, a quota becomes a hard limit.

If a user exceeds the hard limit, a disk quota mechanism in the operating system kernel prevents files from being written to disk, thus compelling the user to remove files.

Any file open in violation of the disk quota cannot be written and its original contents are destroyed. Operations that truncate or remove then rewrite files might fail, because the system refuses to allocate new resources to the offending user. Such operations might result in loss of data. For this reason, users should be warned to heed system warnings when they exceed quotas and take corrrective action as soon as possible, when the system warns them they have exceeded quota.

How Disk Quotas Work

The following diagram shows the model by which disk quotas are maintained:



LG200172_009

Figure 14-10. Disk Quotas Flow Chart

14 All disk-quota commands work via the system call *quotactl*(2), which communicates with the kernel via two disk-quota data structures:

- `struct dqblk` data structure, defined in `quota.h`
- `struct mount` mount tables defined in `mount.h`. The mount table has a flag indicating whether quotas are enabled.

The *quotactl*(2) system call commands are used to update both the data structures and the `quotas` file with mappings of each file system. The `quotas` file retains information on disk-quota settings and utilization statistics for the operating system.

When file systems are mounted, the kernel sets up internal tables in memory containing statistics on usage and limits. The kernel uses these internal tables to maintain information about disk resources allocated or released for users with quotas. The information is flushed to the `quotas` files on each file system whenever the file system is unmounted (such as during a shutdown), to prevent losing the information.

Disk-Quotas Data File—quotas

A `quotas` file, created at the root of each affected file system, stores limits and utilization statistics for each user for the entire file system. To enable disk quotas, you must create a `quotas` file. It is not an ASCII file, but can be accessed in ASCII form by using the `edquota` command.

Each user for whom quotas are being established in a file system must be specified as an entry in the `quotas` file. The `edquota` command allows you to specify individual disk-quota values for users. It also lets you establish template values for a prototypical user, and copying those values to selected users on the system.

The `quotas` file can be written to under the following circumstances:

- when edited with *edquota*(1M)
- when a file system is unmounted
- when `quotacheck` writes to disk

Information is stored in `quotas` according to the elements of the `dqblk` data structure, defined in `quotas.h`. These include:

- block limit (limit)—the maximum allowable number of disk blocks.
- block quota (quota)—the preferred limit of disk blocks.
- block usage—current block count being consumed.
- file limit—maximum number of allocated files.
- file quota—preferred file limit.
- file usage—current number of allocated files.
- time limit for excessive block use.
- time limit for excessive files.

The `quotas` file is read by *repquota*(1M) for current quota statistics. It is written (updated) by `quotacheck` and whenever the file system is unmounted.

How Disk Quotas Appear to Users

As long as they do not exceed allocated disk quotas, users will not notice the presence or enforcement of disk quotas. However, upon logging in, `login` alerts any user who exceeds the allocated disk quotas of this fact.

Users can execute the *quota*(1) command to display their own disk usage and limits. If they exceed disk allocation, they need to remove files to conform to disk-quota requirements.

Superusers can use *quota*(1) to view the limits of any user or *repquota*(1M) to report on entire file systems.

Disk Quota Commands

A variety of commands are available to the system administrator for managing disk quotas. These commands fall into several categories:

- enabling and suspending disk quotas
- maintaining file-system consistency while implementing disk quotas
- setting and editing user quotas
- displaying and reporting on disk quotas

Enabling and Suspending Disk Quotas—quotaon and quotaoff

quotaon(1M) activates disk quotas on one or more file systems. Its syntax is:

`/etc/quotaon [-v ]-a`

14

`/etc/quotaon [-v ]`*file_sys* ...

quotaoff(1M) suspends enforcement of disk quotas on one or more file systems. Its syntax is:

`/etc/quotaoff [-v ]-a`

`/etc/quotaoff [-v ]`*file_sys* ...

The following options and arguments apply to both `quotaon` and `quotaoff`:

`-v`	Produce verbose output.
`-a`	Enable or suspend disk quotas on all mounted file systems cited with the `quota` option in `/etc/checklist`.
file_sys	Enforce or suspend disk quotas on specified mounted file systems. The *file_sys* argument is the name of either the directory on which the file system is mounted or the block special file of the file system.
	Either `-a` or *file_sys* must be specified.

Both `quotaon` and `quotaoff` work by calling *quotactl*(2). `quotaon` and `quotaoff` update the status field of devices located in `/etc/mnttab` to indicate whether quotas are being enforced.

Maintaining File-system Consistency—quotacheck

quotacheck(1M) is used to maintain usage statistics on a file system with disk quotas. Each file system being checked must have a `quotas` file at its root.

`Quotacheck` examines each specified file system and builds a disk-usage table of user IDs and blocks used, which it compares to a table stored in that file system's `quotas` file. If inconsistencies are detected, both the `quotas` file and the kernel quotas table are updated. (Inconsistencies typically develop if a system mounted with disk quotas has been used for a period of time with quotas enforcement suspended.)

`Quotacheck` is executed at various times:

- During a reboot, *quotacheck*(1M) can be run from `/etc/bcheckrc`, before activating disk quotas with `quotaon`.
- When interactively mounting a file system assigned quotas, `quotacheck` should be run after the mount.

14

`/etc/quotacheck` [`-v`]`-a`

`/etc/quotacheck` [`-v`]*file_sys* ...

Like `quotaon` and `quotaoff`, `quotacheck` uses a `-v` (verbose) option, `-a` option instructing the system to check all mounted file systems listed in `/etc/checklist`, and *file_sys* argument, for checking specific mounted file systems.

Since `quotacheck` accesses the raw device in calculating the actual disk usage for each user, the file systems checked should be inactive when `quotacheck` is executed. This is most efficiently done in single-user mode.

Setting and Editing User Quotas—edquota

edquota(1M) is the command used to edit the `quotas` file, something you must do to set disk quotas. Its syntax for editing user quotas is:

`/etc/edquota` [`-p` *proto-user*]*user* ...

`/etc/edquota -t`

The first syntax line shows two ways of specifying quotas for a user:

- Use the *user* argument to specify disk quotas for one or more users.
- Use the -p *proto-user* option with the *user* argument to copy disk quotas from a prototypical user to one or more users.

When invoked, edquota creates a temporary ASCII file with a one-line representation for each user specified on the command line, showing the current disk quotas set for that user. You can then modify or add quotas to the file using an editor (environ variable EDITOR, or vi by default).

A temporary file created by edquota for editing a new user's disk quotas might look like this:

```
fs /mnt blocks (soft=100, hard=120) inodes (soft=0, hard=0)
```

If the quota command is invoked for that user, the following is displayed:

```
$ quota -v user1

Disk quotas for user1 (uid 101):
Filesystem     usage  quota  limit    timeleft  files  quota  limit    timeleft
/users    0     100    120                  0      0      0
```

14

Editing Disk-Quota Time Limits. Time limits are used to enforce disk quotas. Enforcement changes from warning message to restriction after a specified time limit. The syntax for specifying the time limit for a given file system is:

```
/etc/edquota -t
```

When invoked, edquota -t creates an ASCII representation of the quotas file much like this:

```
fs / blocks time limit =0 (default), files time limit =0 (default)
```

The default time limit is defined in quota.h as one week. You can specify the time limit in terms of seconds, hours, days, weeks, and months.

If a user reaches the quota (soft limit), the system generates a warning, begins to keep track of how much time the user exceeds quota, but allows the user to continue working.

The quota output for the user might then look like this:

```
$ quota -v user1

Disk quotas for user1 (uid 101):
Filesystem     usage  quota  limit    timeleft  files  quota  limit    timeleft
/users           114    100    120     4 days     40      0      0
```

However, if the user exceeds the time limit by ignoring warnings to remove files or reaches the limit by creating more files or blocks than alloted, the system prevents the user from writing files. In fact, any file the user attempts to open and write might be destroyed and the following message (in this case, by friendly *vi*(1)) is displayed:

```
DISK LIMIT REACHED - WRITE FAILED (/users)

Your edited changes will be lost if you do not complete
a successful write command before exiting.  If you were
writing out to your original file, the previous contents
of that file have been destroyed.  Contact your System
Administrator BEFORE exiting if you need assistance.

[Hit return to continue.]
```

The user is thus compelled to remove files to below the assigned disk quota.

One thing to bear in mind (and inform users on your system): do not panic when you get a "WRITE FAILED" message. If you work in a windows environment, open up another window. If you are not running X-Windows or some other window manager, suspend the job. From the shell, remove some unnecessary files, then return to the process. The process is still in memory, and once you return to it, if you have brought the number of files or blocks below quota, you will then be able to save the file.

Displaying Users' Disk Quotas—quota

The single most-commonly used disk-quota command is *quota*(1), because it can be used by all users to stay abreast of their own disk-usage status.

`Quota` displays a user's disk quota and usage. Used without options, the command displays warnings about mounted file systems where usage exceeds quota. Remotely mounted file systems (NFS) for which disk quotas are enforced are reported also. (See section on `rquotad` below for more information.) The `-v` option shows all current statistics for the user. For a new user with disk quotas, the output for the `quota` command might look like this:

```
$ quota -v

Disk quotas for user1 (uid 101):
Filesystem     usage  quota  limit    timeleft  files  quota limit    timeleft
/users             0    100    120                  0      0    0
```

As the user creates files, the number of blocks and files the user owns is shown in the usage and files columns.

14

Enforcing Disk Quotas Remotely—rquotad

If the user has quotas on remotely mounted (NFS) file systems, `quota` can report them using the remote quota server daemon, `rquotad`. `/usr/etc/rpc.rquotad` is invoked remotely via *inetd*(1M).

The syntax of `rquotad` is:

```
/usr/etc/rpc.rquotad
```

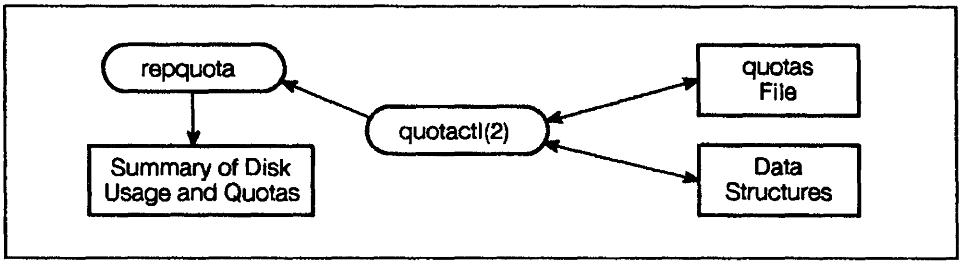
Manipulating Disk Quotas Programmatically—quotactl

quotactl(2) is a system call that manipulates disk quotas. Using *quotactl*(2), you can:

- Turn quotas on for a file system.
- Turn quotas off for a file system.
- Retrieve quota usage information.
- Change quota usage information.
- Set file and block quotas and limits.
- Flush quota cache information to disk (`quotas`)

Summarizing Disk Usage and Quotas—repquota

The *repquota*(1M) command is used to read the kernel's internal disk-quota tables and print a summary of disk usage and quotas for specified file systems. The summary shows the current number of files, amount of space (in kilobytes); and assigned quotas for each user.



LG200172_010

Figure 14-11. How repquota Generates Disk-quota Reports

In the following sample summary, `repquota` is invoked, using the `-v` verbose option, to report on two specific file systems (`/` and `/users`). The `-a` option can be used instead to read all mounted file systems in `/etc/checklist`:

```
$ repquota -v / /users

/dev/root (/):
                      Block limits                         File limits
User     used     soft     hard   timeleft     used    soft    hard     timeleft
julie      20      100     2000                   4      50     400
fred      400      100     2000     50 min       20      50     400


/dev/dsk/1s0 (/users):
                      Block limits                         File limits
User     used     soft     hard   timeleft     used    soft    hard     timeleft
bill      593     1000     1200                 192     500     700
julie     947     1000     1200                 700     500     700     NOT STARTED
fred     4150     3000     6200     4 days      800    1000    1500
```

The output shows three users, `bill`, `julie`, and `fred`. The system is set up so that they can create a small number of files in the root directory, but are allotted more space in the `/users` file system. `julie` and `fred` have files in the root directory as well as in `/users`.

User fred has exceeded his quota (soft limit) of 1K blocks allocated on both / and /users. login warns him of the condition whenever he logs in, but having neglected to heed warnings and decrease block numbers below acceptable levels, fred now has only 50 minutes before the soft limit is enforced as a hard limit in the / directory. He faces the same restriction in /users if he fails to cut his block usage below the 3000K-block quota.

Once time runs out, the timeleft field will read EXPIRED. The kernel will allow fred to log in, but the login program will report to him that he has exceeded his quota. Any login process that creates space on the disk will fail. Any command that he runs that creates new blocks will fail. All he will be able to do is delete files or the contents of files, until he has brought his account below quota.

14 User julie faces a similar problem. Although she has not exceeded her block allocation, she has not only exceeded her quota for number of files, but has reached the hard limit of number of permissible files. Once she exceeded quota, she too received login warnings. Having reached the hard limit, she is unable to create files on /users.

The timeleft field shows julie's transgression somewhat cryptically with the notation "NOT STARTED". When a user reaches allotted hard limits, whether block or file, time is irrelevant. The system no longer permits the user to do anything other than comply with disk quotas.

Glossary

address

In the context of peripheral devices, a set of values that specify the location of an I/O device to the computer. On the Series 300/400, the address is composed of up to four elements: select code, bus address, unit number and volume number. On the Series 800, the address is composed of up to five elements: bus converter, module number, slot address, device switch setting, and logical unit number. Addresses on the Series 700 are composed of device, module number, and switch setting.

Addresses are covered in more detail in Chapter 11, "System Configuration" and the manual *Installing Peripherals*.

advisory lock

A file lock placed on a file to inform (warn) other processes wanting to access the file that it is already being accessed and potentially being modified. Advisory locks are useful only among cooperating processes that use file locking. (See Chapter 8, "HFS File System".)

access

An interaction between a subject and object resulting in information flowing from one to the other.

access control list (ACL)

A set of entries associated with a file that specify permissions for all possible user-ID/group-ID combinations. (See *acl*(5) in the *HP-UX Reference* and *HP-UX System Security*.)

access control mechanism (ACM)

An algorithm and data structure that supports access control decisions. It mediates decisions about whether specific subjects can access specific system objects in specific ways. An ACM might be implicit (contextual),e.g. “only user ID 0 can access files in /etc”. It might be explicit, as in a list of objects and their access rights. An explicit ACM might be distributed (data stored with protected objects), as in the UNIX file permissions scheme, or it might be centralized, as in a file describing access rights for all protected system objects.

allocation policy
Allocation policy governs how disk space is distributed. When using the Logical Volume Manager (LVM), the allocation policy governs how disk space is distributed to logical volumes and how extents are laid out on an LVM disk.

(See Chapter 9, “Logical Volume Manager”.)

archived library
Object files that are linked to a program when called, regardless of whether another copy of that library already exists in memory. Compare with **shared library**.

ARPA hostname
Needed for ARPA Services. A name consisting of one field containing any printable character except spaces, newlines, or the pound (See NS_ARPA Services documentation.) sign (#). Assigned in `/etc/hosts`.

attended mode
Occurs when the user selects an operating system during system startup, rather than letting the boot ROM automatically find an operating system. (See Chapter 2, “System Startup”.)

attribute
In an HP-UX cluster, the attributes are characteristics of the cluster nodes, such as the cluster node name or processor type. The attributes are collectively referred to as the cluster node's `context`. (See *Managing Clusters of HP 9000 Systems*.)

audit trail
A set of records that collectively provide documentary evidence of processing, used to aid in tracing from original transactions forward to related records and reports, and/or backwards from records and reports to their component source transactions.

audit ID

An ID associated with each user when trusted systems are used. The audit ID does not change, even when executing programs with a different user ID. (See Chapter 5, "Process Management" and *HP-UX System Security*.)

autocreation

If you create a CDF, but specify only the path name of the CDF and not a subfile, the system automatically creates a CDF subfile named after the cluster node name attribute. This is known as autocreation. (See *Managing Clusters of HP 9000 Systems*.)

available memory

Memory is used by the system for executing user processes. It is the amount of physical memory (main memory) on your system not reserved by the HP-UX kernel for device drivers, system data structures, and other kernel needs. (See Chapter 7, "Memory Management".)

background process group

A process group that is currently not running as the foreground process group. Typically, only one process group at a time is the foreground process group in a session, while all other process groups are background process groups.

bad block relocation

A feature of LVM for recovering corrupted blocks of data from HP-FL and SCSI disks, by redirecting the data to another block of media on the disk. (See Chapter 9, "Logical Volume Manager".)

binding

The process by which the operating system's self-configurating capabilities find hardware on the bus architecture and associate it with relevant software (device files and device drivers). (See Chapter 11, "System Configuration".)

bit

The smallest unit of information in a digital computer. Its value can be either 0 or 1. Bits are normally grouped together in bytes and words.

bits per inch

A common measure of "tape density." Indicates how many bits can be stored per inch of magnetic tape. (See Chapter 12, "HP-UX Peripherals".)

block

The fundamental unit of information HP-UX uses for access and storage allocation on a mass storage medium. Block size on an HFS file system can be either 4 KB or 8 KB, and is set at file system creation. The default block size is 8 KB. However, to present a more uniform interface to the user, most system calls and utilities use "block" to mean 512 bytes, independent of the actual block size of the medium. (See Chapter 11, "System Configuration".)

block device

A hardware device that transmits and receives data in blocks. (See Chapter 11, "System Configuration".)

block mode

Also termed buffered I/O. Data is held in the buffer cache, then transferred one block at a time. Block size for buffered I/O is not the same as block size on the file system. Block size for buffered I/O is defined as BLKDEV_IOSIZE in `/usr/include/sys/param.h`. Compare with **raw mode**. (See Chapter 11, "System Configuration".)

boot or boot-up

The process of loading, initializing and running an operating system. (See Chapter 2, "System Startup".)

boot area

Eight kilobytes of the disk that are reserved for system use. This area contains the LIF volume header, the directory that defines the contents of the volume, and the bootstrapping program. On the Series 300 and 400, the 8 KB are at the beginning of the disk; on the Series 700, they are at the end of the disk; and on a standard implementation of Series 800, they are in their own section (typically section 6) or logical volume. On a Series 800 implementing either LVM for the root file system or the SwitchOver/UX product, the LABEL file is stored in the boot area. (See Chapter 8, "HFS File System".)

boot process

A process that runs during the HP-UX phase of system startup. For example, `/etc/rc` and `/etc/brc` are boot processes. (See Chapter 2, "System Startup".)

boot ROM

The **boot ROM** is small, machine language program that resides in your computer's read-only memory. When you boot or re-boot the system, the computer starts the boot ROM program which takes control of the system. The boot ROM tests computer hardware, finds some devices accessible through the computer, and loads an operating system. (See Chapter 2, "System Startup" in this manual or Chapter 5, "System Boot-Up Problems" in *Solving HP-UX Problems*.)

bpi

See **bits per inch**.

buffer cache

A buffer which remains in memory and which is used by the file system for buffering writes to the file system. (See Chapter 8, "HFS File System".)

bus

Hardware that carries data and signals between hardware modules within the system. Typically, a bus is implemented as soldered parallel wires either on a frame to which boards are plugged, on a board housing multiple chips, or internal to a processor for transmission to and from memory, other processors, or other buses.

bus address

Part of an address used for devices (such as HP-IB, SCSI, EISA, or HP-FL); a number determined by the switch setting on a peripheral that allows the computer to distinguish between two devices connected to the same interface. A bus address is sometimes called a device address; no two devices on the same HP-IB can have the same bus address. (See Chapter 11, "System Configuration".)

bus converter

A board that serves as an internal interface between higher- and lower-speed bus, typically the SMB and Mid-Bus. The bus converter handles timing and bus protocol management. Because more than one can be connected to the SMB, bus converters can also increase the number of modules in a system, thus enabling more devices to be connected.

byte

A basic unit of digital computer information, consisting of 8 consecutive bits.

byte offset

On the Series 800, a 32-bit quantity that points to the specific location of the virtual address space being accessed. (See Chapter 7, "Memory Management".)

bytes per inode

This specifies the number of data bytes (amount of user file space) per inode slot. The number of inodes is calculated as a function of the file system size. The default value is 2048. (See Chapter 8, "HFS File System".)

cache

A small high-speed memory between RAM (main memory) and the processor that may hold data and text for active processes. Cache designed into computer-system architecture accelerates effective memory transfer rates and processor speed. If the data is not in cache, the processor consults the translation tables in RAM that hold the mapping between virtual and physical addresses of the data. (See Chapter 7, "Memory Management".)

central processing unit (CPU)

The instruction-processing module inside the computer, typically including the processor, floating-point co-processor, and special registers. On multi-processing computers (computers with more than one processor), the term system processing unit (SPU) is used.

certification

The technical evaluation of a system's security features, made as part of and in support of the approval/accreditation process, that establishes the extent to which a particular computer system's design and implementation meet a set of specified security requirements.

CDF

Context-dependent file. The mechanism used by HP-UX clusters to share a path name between different cluster nodes in a cluster. A CDF consists of a hidden directory and one or more subfiles. (See *Managing Clusters of HP 9000 Systems*.)

channel adapter manager (CAM)

On the Series 800, the type of device driver (`cio_ca0`) for the CIO bus that provides the software interface to device adapters (such as `hpib0`, `mux0`, and `scsi2`). The channel adapter manager manipulates data between the memory mapped I/O and CIO bus.

channel I/O adapter

On the Series 800, a chip on a board or one or several Mid-Bus cards that serve as an internal interface between the Mid-Bus and the lower-speed CIO bus. The channel adapter synchronizes the different speeds and bandwidths of the Mid-Bus and CIO bus. The channel adapter also manages direct transfer of data to and from main memory with minimal CPU intervention, and by extension, to the rest of the I/O system. More than one channel adapter can be installed on a system, thus expanding the number of possible modules.

character device

Typically, a device which is not a block device and which transfers data a character a time. Printers, plotters, terminals, magnetic tape drives, and pseudo devices are all examples of character devices. (See Chapter 11, “System Configuration”.)

clean byte

A flag in the primary superblock which the `fsck` command uses to determine whether the file system state is consistent. (See Chapter 2, “System Startup” and Chapter 8, “HFS File System”.)

cluster

An HP-UX cluster is one or more workstations linked together with a local area network (LAN), but consisting of only one file system. HP-UX clusters are supported on Series 300/400/700 only. (See *Managing Clusters of HP 9000 Computers*.)

cluster client
A cluster node that does not have the root file system for the cluster on its local disks. Its file system resides on the cluster server. However, cluster clients can have locally mounted disks for local data storage. (See *Managing Clusters of HP 9000 Computers*.)

cluster node
A computer in a cluster. (See *Managing Clusters of HP 9000 Computers*.)

cluster server
The cluster node that acts as a file-system server for all the clients in an HP-UX cluster. (See *Managing Clusters of HP 9000 Computers*.)

cluster server process (CSP)
A special kernel process that runs on a machine in a cluster to satisfy requests from other nodes in the cluster. There are two kinds of CSP, the limited CSP (LCSP) and the general CSP (GCSP). (See *Managing Clusters of HP 9000 Computers*.)

code segment
Glossary
The memory in segment into which a process's executable code is loaded by `exec`. Same as the **text segment**. (See Chapter 7, "Memory Management".)

comment lines
In the context of system accounting, lines in the `holidays` file that begin with an asterisk (*) in the first column. (See Chapter 14, "System Accounting".)

company holiday lines
Used in the system accounting file, `/usr/lib/acct/holidays`, to denote holidays. (See Chapter 14, "System Accounting".)

configuring
Telling the operating system what physical hardware is present, by associating the hardware with necessary software—device drivers and device files. (See Chapter 11, "System Configuration".)

connect session
This denotes the period of time in which a user is connected to the system.

It starts when the user logs in and finishes when the user logs out. (See Chapter 14, "System Accounting".)

context

The mode (either system or user) a process is in when executing. (See Chapter 5, "Process Management".)

In HP-UX clusters, context is an ASCII string comprised of a number of attributes that determine which subfile (if any) is accessed from a CDF. Each workstation on a cluster has an associated context set at boot time. The context attributes include cluster-node name; processor type; floating point hardware; file system location; and the string "default". (See *Managing Clusters of HP 9000 Computers*.)

context-dependent file

A file in a cluster having different contents, depending on which cluster node uses it. A context-dependent file consists of a hidden file and one or more subfiles. Also called a **CDF**. (See *Managing Clusters of HP 9000 Computers*.)

controlling process

The session leader that connected to the controlling terminal. (See Chapter 5, "Process Management".)

controlling terminal

Every session has exactly one controlling terminal, which all processes in that session, by default, use for standard input, standard output, and standard error. Every controlling terminal is associated with, at most, one session. (See Chapter 5, "Process Management".)

copy-on-write

On Series 300/400, when a process forks, the parent process plans a duplicate image of itself at child virtual addresses. But instead of actually copying the image, the parent pages are marked "copy-on-write." The pages continue to reside in the parent address space, and are not copied until the process actually writes on the page. On Series 700/800, HP-UX implements a variation of copy-on-write, called "copy-on-access." (See Chapter 7, "Memory Management".)

cron

This process wakes up every minute to execute commands at specified dates and times, according to instructions in files contained in the directory `/usr/spool/cron/crontabs`. See the *cron*(1M) and *crontab*(1) entries in the *HP-UX Reference* for more details.

CS/80

A family of mass storage devices that communicate with a computer via the CS/80 (Command Set '80) or SS/80 (Sub Set '80) command set. In HP-UX your file system can be on either a SCSI drive or a CS/80 drive.

cylinder

One or more vertical collections of tracks in a disk pack. Disk accesses within a cylinder do not need a seek. (See Chapter 8, "HFS File System".)

cylinder group

One or more consecutive cylinders. Each cylinder group contains a superblock, inodes, cylinder group information, and data blocks. (See Chapter 8, "HFS File System".)

cylinder group information

A data structure located in the cylinder group that contains information about the cylinder group such as numbers of available inodes, data blocks, and fragments, and bitmaps to free space in the cylinder group. (See Chapter 8, "HFS File System".)

daemon

Background process that performs systemwide functions. For example, `pagedaemon`, `vhand`, and `swapper` are daemons that are part of the HP-UX memory-management subsystem.

daily command summary

A report produced by system accounting, which summarizes command usage for the day. (See Chapter 14, "System Accounting".)

daily line usage report

A report produced by system accounting, which summarizes tty line usage for the day. (See Chapter 14, "System Accounting".)

daily resource usage report

A report produced by system accounting, which summarizes resource (e.g., disk, memory, cpu) usage for the day. (See Chapter 14, "System Accounting".)

DASS

Direct Access Secondary Storage. A form of mass storage with characteristics of both online magnetic disk primary storage and offline magnetic tape secondary storage. (See Chapter 12, "HP-UX Peripherals" in this manual and *autochanger*(7) in the *HP-UX Reference Manual*.)

data segment

A memory segment into which a process's static and dynamic data is stored. (See Chapter 5, "Process Management".)

demand paging

A form of virtual memory management in which pages are brought into memory only as they are accessed. At any one time, only a subset of the process need be loaded into main memory. (See Chapter 7, "Memory Management".)

demand-loadable

Even though HP-UX is a demand-paged virtual memory system, it still attempts (by default) to load an entire process into main memory. If, instead, code is marked as demand-loadable, then the process's code will be brought into physical memory only as pages are accessed.

demand-paged virtual memory

This is the type of memory management implemented on Series 300 and 800 computers. See also **demand paging**.

destination device

The mass storage device on which HP-UX is to be installed or updated. The destination device must be a hard disk drive.

device adapter

Interface cards (such as HP-IB, HP-FL, SCSI, LAN, and MUX cards) installed in the I/O slots of the CIO, HP-PB, EISA, or other bus. Device adapters make the crucial link that enables the system to communicate with external peripherals (such as terminals, printers, disk drives). Device adapters are frequently called I/O cards. CIO device adapters are also called CIO cards; HP-PB device adapters are called HP-PB cards. (See Chapter 11, "System Configuration" and *Installing Peripherals*.)

device adapter manager (DAM)
On the Series 800, the type of device driver that deals with the device-adapter protocol. For example, `hpib0` handles HP-IB data protocol transfer for the CIO bus; `hpib1` handles HP-IB data protocol transfer for the HP-PB bus. (See Chapter 11, "System Configuration".)

device manager (DM)
On the Series 800, the type of device driver (such as `mux`) that translates requests from the CIO bus to the attached interface card. (See Chapter 11, "System Configuration".)

device driver
On the Series 300/400/700, the term used for the driver that controls kernel data structures from the interface card to the external device itself. On the Series 800, device driver is also used more generally to mean the software that enables the operating system to transmit data between the computer and the external device. Series 800 device drivers include channel I/O adapter manager (CAM), device manager (DM), device adapter manager (DAM), and logical device manager (LDM). (See Chapter 11, "System Configuration".)

device file
See **special file.**

device swap space
Swap space that occupies a disk, section, or logical volume of a disk reserved exclusively for swapping. It is fast. Because at least one swap device must be present on a system, device swap space is also referred to as **primary swap space.** (See Chapter 7, "Memory Management".)

Discretionary Access Control (DAC)
A means of restricting access to objects based on the identity of subjects and/or groups to which they belong. The controls are discretionary in the sense that a subject with a certain access permission is capable of passing that permission (perhaps indirectly) on to any other subject (unless restrained by mandatory access control).

disk quotas

A subsystem used to limit the number of files and file blocks a user can own per file system. (See Chapter 14, "System Accounting".)

disk sectioning

On Series 800 systems, disks can be divided into sections, each one appearing to the operating system as if it were a separate disk. This is known as disk sectioning or partitioning. A more flexible Series 800 alternative to disk sectioning is provided by the Logical Volume Manager (LVM). (See Chapter 8, "HFS File System" and Chapter 9, "Logical Volume Manager".)

diskless workstation

Same as a **cluster client**.

driver

Compiled code (supplied with HP-UX), which resides in the kernel and handles input and output with peripheral devices. Also called **device driver**. (See Chapter 11, "System Configuration".)

driver number

A number identifying a device driver. Same as **major number**. (See Chapter 11, "System Configuration".)

disk

Collection of recording platters contained in a single disk drive or disk-drive library.

domain

The set of objects that a subject has the ability to access. A set of (object, rights) pairs. Each pair specifies an object and some subset of the operations that can be performed on it. A right in this context means permission to perform one of the operations.

dynamic swap space

Swap space allocated as needed while the system is running. Compare this with primary swap space. Both device and file-system swap can be added dynamically. (See Chapter 7, "Memory Management".)

effective user ID
A form of user ID that allows users access to files they do not own. (See Chapter 5, "Process Management".)

effective group ID
A form of group ID that allows users access to files not in their group. (See Chapter 5, "Process Management".)

enforcement mode
A form of file locking in which a user is not allowed to read or write to a file while another user is accessing the file. In such a case, the requesting process sleeps until the process that has control of the file releases it. (See Chapter 8, "HFS File System".)

exchange
An exchange occurs when an optical autochanger replaces one disk surface in the optical drive with another. (See Chapter 12, "HP-UX Peripherals".)

extent
Fixed-size addressable areas of space on an LVM disk or in memory. On disk, these areas are called physical extents, and are by default 4 MB. Physical extents map to areas in memory, called logical extents. If disk mirroring (HP MirrorDisk/UX product) is used, each logical extent corresponds (maps) to more than one physical extent. If disk mirroring (HP MirrorDisk/UX product) is *not* used, there is a one-to-one relationship between the two types of extents. (See Chapter 9, "Logical Volume Manager".)

file
A discrete collection of information described by an inode and residing on a mass storage medium.

file locking
Measures which help to ensure that only one user at a time can access a particular file or files. See **enforcement mode** and **advisory lock**. (Also, see Chapter 8, "HFS File System".)

file mark
A special type of record written to tape and recognized as a boolean condition during reading. Single file marks separate records on tape; two

consecutive file marks indicate the end of tape. (See Chapter 12, "HP-UX Peripherals".)

file-system swap space

A secondary form of swap space, enabled by `sam` and set in `/etc/checklist`, that allows a process to use an existing file system if device swap space is insufficient to meet demand-paging needs.

File-system swap is slower than device swap but varies in size with the system's swapping activity. (See Chapter 7, "Memory Management")

file types

The file type is established at the time of the file's creation. The types are:

- Regular files - Contains a stream of bytes. Characters can be either ASCII or non-ASCII. This is generally the type of file a user considers to be a file: object code, text files, nroff files, etc.
- Directory - HP-UX treats directories like regular files, with the exception that writing directly to directories is not allowed. Directories contain information about other files.
- Block special files - Device files that buffer the I/O. Reads and writes to block devices are done in block mode.
- Character special files - Device files that do not buffer the I/O. Reads and writes to character devices are in raw mode.
- Network special files - contain the address of another system.
- Pipes - A temporary file used with command pipelines. When you use a pipeline, HP-UX creates a temporary buffer to store information between the two commands. This buffer is a file, and is called a pipe.
- FIFO - A named pipe. A FIFO (First In/First Out) has a directory entry and allows processes to send data back and forth.
- symbolic link - A type of file that indirectly refers a path name.

file system

The organization of files and directories on a hard disk. The HFS file system is an implementation of the HP-UX directory structure. NFS provides access to a file system over a network. (See Chapter 8, "HFS File System".)

foreground process group
The process group in a session which currently is executing in the foreground and has control of the controlling terminal. (See Chapter 5, "Process Management".)

fragment
A piece of a block. This is the smallest unit of information HFS will read or write. The lower limit of a fragment is DEV_BSIZE (defined in `/usr/include/sys/param.h`). Fragment size is set at file system creation. (See Chapter 8, "HFS File System".)

frame
A single, nine-bit character on a magnetic tape. (See Chapter 12, "HP-UX Peripherals".)

free space threshold
Specifies minimum *percentage* of free disk space allowed. Once the file system capacity reaches this threshold, only the system administrator is allowed to allocate disk blocks. The default is 10%; if it is less, file system performance degrades. The free space threshold is set when you create a new file system. Same as `minfree`. (See Chapter 8, "HFS File System".)

fs_bsize
The size of disk blocks in the file system. (See Chapter 8, "HFS File System".)

generic device file
A device file whose cluster node ID is 0, thus allowing it access by all cluster nodes. (See *Managing Clusters of HP 9000 Systems*; (Series 300/400/700 only.)

gateway page
A Series 800 address range used for switching between user and kernel privileges when making system calls.

group access list
A list of groups whose users are allowed access to a file. (See Chapter 5, "Process Management" and Chapter 8, "HFS File System".)

halting

Bringing the system to a complete stop, a state in which no processes are running. The only way to get HP-UX running again is to cycle power or do a hardware reset. (See Chapter 2, "System Startup".)

hard limit

In disk quotas, a hard limit (also termed limit) is the maximum amount of file space allowed by a given user. (See Chapter 14, "System Accounting".)

hardware path

The sequence of hardware addresses from bus converter (if applicable), to module, to device adapter, to device.

HFS file system

High performance File System, the file system implemented on HP-UX 9000 systems. (See Chapter 8, "HFS File System".)

hidden directory

A directory used to implement a CDF. It is called hidden because it is normally treated and seen as a file. It can be accessed as a directory only by appending the special character "+" to its name. (See *Managing Clusters of HP 9000 Systems*.)

home directory

The directory into which a user is placed after `login`. The user's home directory is defined in the `/etc/passwd` file. (See Chapter 4, "Login".)

homogeneous cluster

An HP-UX cluster containing only systems of the same architecture (for example, all Series 300 and 400 or all Series 700). (See *Managing Clusters of HP 9000 Systems*.)

HP-PB

HP Precision Bus. Refers to the hardware I/O architecture of HP 9000 models such as Models 8x2 and 8x7. (See Chapter 10, "System Architectures".)

HP-PB adapter

See **device adapter**.

HP-UX directory structure
The hierarchical grouping of directories and files on HP-UX. (See Chapter 8, "HFS File System" and *A Beginner's Guide to HP-UX*.)

HP-UX system hostname
Name you assign to your system (from a line on `/etc/rc`). It is used for UUCP, mail, and other programs.

immediate response mode
A mode of data transfer in which data is read or written immediately to mass storage, without buffering up the data for more efficient I/O. (See Chapter 12, "HP-UX Peripherals".)

initdefault
The default run-level for `init` to use during system startup. (See Chapter 2, "System Startup".)

inode
A data structure containing information about a file such as file type, pointers to data, owner, group, and protection information. (See Chapter 8, "HFS File System".)

interface driver
On the Series 300/400/700, the type of device driver that controls the kernel data structures concerning memory-mapped and processor I/O.

Internet address
An address used by NS_ARPA Services. It consists of two parts: a network number and a host number. Nodes on the same LAN will have the same network number and distinct host numbers. (See NS_ARPA Services documentation.)

I/O channel separation
A capability, when using LVM, for segregating highly I/O-intensive areas. For example, you might have a database on one channel and file systems on another. (See Chapter 9, "Logical Volume Manager".)

I/O adapter
See **device adapter**.

I/O mappings
The range of addresses the operating system uses to deal with I/O devices.

ITE
The Internal Terminal Emulator program which allows a bit-mapped display to function as a standard computer terminal.

job
An individual instance of a command. Can also be two or more commands piped together. (See Chapter 5, "Process Management".)

job control
HP-UX commands and key sequences that allow users to manage jobs more effectively. (See Chapter 5, "Process Management", *A Beginner's Guide to Using Shells*, and *Using HP-UX*.)

kernel
The core of the HP-UX operating system. The kernel is the compiled code responsible for managing the computer's resources, such as memory, file system and input/output (I/O). The kernel resides in RAM (Random Access Memory) whenever HP-UX is running. (See Chapter 11, "System Configuration".)

kernel stack segment
Virtual address space that contains the run-time stack when a process is running in kernel mode. (See Chapter 7, "Memory Management".)

LIF
Logical Interchange Format. LIF is Hewlett-Packard's standard file format, used for transferring files between Hewlett-Packard systems. Since LIF is a standard, files with LIF format can easily be transferred between different Hewlett-Packard computers (See Chapter 8, "HFS File System" and *lif*(4) in the *HP-UX Reference*.

logical volume
When using the Logical Volume Manager (LVM), a logical volume is a storage device, much like a disk section but of flexible size, that can hold a file system (including root), raw data, application program, or swap. Logical volumes can be mirrored using an optional product, HP LVM Mirroring.

Because its data is distributed *logically* (rather than physically), a single logical volume can be mapped to one LVM disk or span multiple disks, but the logical volume itself is used as a single virtual disk. (See Chapter 9, “Logical Volume Manager”.)

Logical Volume Manager (LVM)
An operating system software module that implements virtual (logical) disks to extend, mirror, and improve the performance of physical disk access. (See Chapter 9, “Logical Volume Manager”.)

LVM disk
A disk that has been initialized for LVM; also termed a **physical volume**. (See Chapter 9, “Logical Volume Manager”.)

limit
In disk quotas, a limit (or hard limit) is the maximum amount of file space allowed by a given user. (See Chapter 14, “System Accounting”.)

link level address
A unique 12-digit hexadecimal number which is part of every LAN card. This number appears on the LAN card hardware, on the boot ROM screen, and can be obtained using the `landiag` program.

local login script
A `.login` script, contained in a user's home directory, which provides local control over the user's working environment. (See Chapter 4, “Login”.)

local swapping
In an HP-UX cluster, cluster clients can have locally mounted disks to which they can do swapping. This is known as local swapping. (See Chapter 7, “Memory Management” and *Managing Clusters of HP 9000 Systems*.)

logical device manager (LDM)
On the Series 800, the kind of device driver that translates requests between the interface card and the external peripheral. For example, `lpr0`.

login
The process by which user gains access to HP-UX. This process consists of

entering a valid user name and its associated password (if one exists). (See Chapter 4, "Login".)

lockable memory
Memory that can be locked into memory by processes via the `plock` and `shmctl` system calls (described in section 2 of the *HP-UX Reference*). (See Chapter 7, "Memory Management".)

logical unit (`lu`)
The logical unit (`lu`) specification is created by *insf*(1m) to allow one device driver to handle many devices. The `lu` is mapped into a table that the kernel then consults to determine which specific hardware address the driver wants to access. The `lu` allows the system administrator to keep track of all configured devices belonging to the same class of device. (See *insf*(1m) in the *HP-UX Reference* and Chapter 11, "System Configuration".)

magneto-optical (MO)
Magneto-optical is a form of rewritable optical technology. (See Chapter 12, "HP-UX Peripherals".)

major number
An index into a device driver table in the kernel. It is needed to communicate with peripheral devices. (See Chapter 11, "System Configuration".)

manufacturing automation protocol (MAP)
A communications standard enabling HP-UX to control factory-floor equipment, including robotics. (See Chapter 11, "System Configuration".)

mechanical changer
The part of the optical autochanger that moves optical disks from slot to drive and vise versa. (See Chapter 12, "HP-UX Peripherals".)

mechanical picker
See **mechanical changer**.

media
In terms of optical products, this is the optical disk that holds the data.

The term includes the plastic cartridge that houses the optical disk. (See Chapter 12, "HP-UX Peripherals".)

memory management unit (MMU)
A hardware component that maps logical address to physical addresses in the virtual memory system. Also protects against illegal accesses by processes into each others' address space. (See Chapter 7, "Memory Management".)

minfree
See **free space threshold**.

minor number
Part of a device file; a hexadecimal number made up of driver-specific information. The minor number often contains device addressing information such as SCSI address, multiplexer port number, or select code specific to the peripheral device you are setting up. (See Chapter 11, "System Configuration".)

mirroring

Simultaneous replication of data using an optional product, HP MirrorDisk/UX. This capability ensures a greater degree of data availability. Mirroring maps logical volumes to multiple LVM disks, thus providing the means to recover easily from the loss of one copy (or two copies in the case of three-way mirroring) of data. Mirroring can provide faster access to data for database applications using more data reads than writes. (See Chapter 9, "Logical Volume Manager".)

monthly total command summary
A report produced by system accounting summarizing command activity for the month. (See Chapter 14, "System Accounting".)

multiprocessing (MP)
Computer processing using two, three, or four system processing units (SPUs), with one source of memory shared among the processors and peripherals. Multiprocessing is a limited form of parallel processing. (See Chapter 5, "Process Management".)

multi-user run-level

A run-level of HP-UX when terminals in addition to the system console allow communication between the system and its users. The multi-user run-level is run-level 2 as shipped. (See Chapter 6, "Run-Levels".)

MUX

MUX is an abbreviation for Asynchronous Multiplexer. Each channel is an RS-232C port which is normally associated with a `/dev/ttyXX` file.

named object

Objects which have names, are visible at the TCB interface, and are shared among users.

NCSC

National Computer Security Center, the government agency that wrote the guidelines for trusted systems.

node

A computer on a network.

NS_ARPA Services

A combined networking product providing both NS and ARPA services. The NS_ARPA Services networking product enables your Series 300 or 800 to transfer files to and from remote hosts, log into remote hosts, execute commands on remote hosts, and send mail to and receive mail from remote hosts on the network. (See NS_ARPA Services documentation.)

NS nodename

Needed for Network Services and the `rlb` daemon. A name consisting of three fields separated by periods, i.e. node.domain.organization. Each field can contain up to 16 alphanumeric, case insensitive characters.

objects

Passive entities that contain or receive information. Access to an object potentially implies access to the information it contains. Examples of objects are: records, blocks, pages, segments, files, directories, directory trees, and programs, as well as bits, bytes, words, fields, processors, video displays, keyboards, clocks, printers, networks nodes, etc.

optical autochanger
A rewritable optical mass storage peripheral which includes the mechanics to move optical disks in and out of drive(s), the drive(s), media and controller electronics. (See Chapter 12, “HP-UX Peripherals”.)

orphaned process group
A process group in which the parent process of every member is either itself a member of the group or is not a member of the group's session. (See Chapter 5, “Process Management”.)

page fault
When the process encounters an address for which no virtual-to-physical translation currently exists on the system, the system issues a page fault. The kernel then switches execution to kernel mode to locate the sought-after virtual address. (See Chapter 7, “Memory Management”.)

page
The smallest contiguous block of physical memory that can be allocated for storing data and processes. On all HP-UX systems, a page is 4 KB in size. (See Chapter 7, “Memory Management”.)

paging
The means by which the memory-management subsystem moves pages of data between RAM and mass storage for execution as a process demands them. (See Chapter 7, “Memory Management”.)

parent process
The parent of a process—that is, the process that spawned a process. (See Chapter 5, “Process Management”.)

parent process ID
The process ID of a process's parent. (See Chapter 5, “Process Management”.)

password
A private character string that is used to authenticate a username to HP-UX. (See Chapter 4, “Login”.)

path name

A series of directory names separated by / characters, and ending in a directory name or a file name. (See *A Beginner's Guide to HP-UX*.)

peripheral device

Any input/output device that can be connected to your system. Disk drives, tape drives, printers, plotters, and terminals are all examples of peripheral devices. (See Chapter 11, "System Configuration", Chapter 12, "HP-UX Peripherals" and *Installing Peripherals*.)

peripheral location

A peripheral's hardware address (minor number). (See Chapter 11, "System Configuration".)

per-process region (`pregion`)

A logical segment that points to specific segments of a process, including text (process instructions), data, u_area and kernel stack, user stack, one or more shared-memory segments, and shared-library text and data segments. `Pregions` hold page protections and the number of pages mapped to each segment. (See Chapter 7, "Memory Management".)

phantom record

A record that crosses the EOT mark on a magnetic tape. (See Chapter 12, "HP-UX Peripherals".)

physical memory

The actual hardware memory components contained within your computer system. (See Chapter 7, "Memory Management".)

physical volume

A disk that has been initialized by LVM for use in a volume group; an LVM disk. (See Chapter 9, "Logical Volume Manager".)

physical volume group

When using LVM, physical volumes (LVM disks) can be further organized within a volume group into a physical volume group, on the basis of the I/O channel or interface adapter to which they are connected, to achieve higher availability of mirrored data. (See Chapter 9, "Logical Volume Manager".)

PID

Same as **process ID**.

port

The place of access through which I/O comes into contact with the CPU.

PPID

Same as **parent process ID**.

primary storage

Typically consists of fixed hard disk(s), used for fast, random-access applications. The primary storage devices are used as online system disks. (See Chapter 8, "HFS File System".)

primary swap space

A contiguous area of a hard disk reserved for use by the demand-paged virtual memory system for swapping. The size and location of primary swap remain fixed in size and location, but can be changed either by reconfiguring the kernel or dynamically, while the system is running. (See **device swap space** and Chapter 7, "Memory Management".)

priority

Ranking of processes by relative importance; used by the scheduler in determining which process the CPU executes next. (See Chapter 5, "Process Management".)

process

A process is the environment in which a program (or command) executes. It includes the program's code, data, status of open files, value of variables, and more. For example, whenever you execute an HP-UX command, you are creating a process; whenever you log in, you create a process. (See Chapter 5, "Process Management".)

process group

When a user spawns a job, HP-UX assigns all processes in the job to a process group. Signals can propagate through all processes in the group. Each process group has a **process group ID** and a **process group leader**. (See Chapter 5, "Process Management".)

process group ID
An integer number which identifies a process group. It is the same as the process ID of the process group leader. (See Chapter 5, "Process Management".)

process group leader
Assigned by HP-UX when a process group is created. The PID of the process group leader is the same as the process group ID of the process group. (See Chapter 5, "Process Management".)

process group lifetime
A period of time beginning when a process group is created and ending when the last remaining process in a group leaves, caused by either:

- The proces lifetime ends, or
- The process calling the `setsid` or `setpgid` system calls (defined in section 2 of the *HP-UX Reference*.

(See Chapter 5, "Process Management".)

process ID
An integer, assigned to a process at creation, which uniquely identifies the process to HP-UX. That is, no two existing processes can have the same process ID. (See Chapter 5, "Process Management".)

process lifetime
After a process is created with a `fork()` function, it is considered active. Its thread of control and address space exists until it terminates. It then enters an inactive state where certain resources may be returned to the system, although some resources, such as the process ID, are still in use. When another process executes a `wait()` or `waitpid()` function for an inactive process, the remaining resources are returned to the system. The last resource to be returned to the system is the process ID. At this time, the lifetime of the process ends.

pseudo-swap reservation
On Series 800, a variation on traditional swapping that allows users to execute processes in memory without allocating physical swap. (See Chapter 7, "Memory Management".)

quorum

When using LVM, quorum is the requirement that a volume group have at least half the configured LVM disks present to change or activate that volume group. If there is no quorum, LVM prevents the change. Quorum is also required to maintain Mirror Write Cache. (See Chapter 9, "Logical Volume Manager".)

quota

In disk quotas, a quota (also termed soft limit) is the amount of file space allocated for a given user, and can be exceeded temporarily. (See Chapter 14, "System Accounting".)

random access memory (RAM)

Memory cards that plug into the computer's backplane. For the CPU to execute a process, the entire process must exist in RAM. (See Chapter 7, "Memory Management" and Chapter 11, "System Configuration".)

raw mode

Unbuffered I/O. Data is transferred directly between the device and the user program requesting the I/O, rather than going through the file system buffer cache. Also known as character mode. Compare with **block mode**. (See Chapter 11, "System Configuration".)

rc command

This is the system initialization shell script `/etc/rc`. The actions that it performs depend on the state in which it is invoked. (See Chapter 2, "System Startup".)

real user ID

An integer, assigned to a username at login from the `/etc/passwd` file, which uniquely identifies the username to HP-UX.

real group ID

An integer, assigned to a username at login from the `/etc/passwd` file, which identifies what group the user is from. Group IDs are defined in the file `/etc/group`.

reboot

Taking HP-UX from a running state, down to a stopped state, and back to a running state. (See Chapter 3, "System Shutdown".)

record

A group of related data items, usually stored contiguously on a mass storage medium. (See Chapter 12, "HP-UX Peripherals".)

region

Data structures unique to a file or chunk of memory being accessed. Regions inform the process about where the data exists in physical memory. (See Chapter 7, "Memory Management".)

remote swapping

The default mode of swapping in an HP-UX cluster. That is, the swapping is performed for a cluster node on the cluster server. (See Chapter 7, "Memory Management" and *Managing Clusters of HP 9000 Systems*.)

rewritable optical

An optical disk technology which can be repeatedly written. (See Chapter 12, "HP-UX Peripherals".)

resource

Anything used or consumed while performing a function. The categories of resources are: time, information, objects (information containers), or processors (the ability to use information). Specific examples are CPU time; terminal connect time; amount of directly-addressable memory; disk space; number of I/O requests per minute.

read-only memory (ROM)

Memory that can be read from (for example, the boot ROM resides in such memory), but cannot be written to. It retains its state (memory) even after power is shut off.

root

Root refers to the highest level directory (root directory or /).

root file system

The file system containing the HP-UX kernel. The kernel is loaded from the root file system at system startup. (See Chapter 2, "System Startup".)

root logical volume

When using LVM, the root logical volume is the logical volume containing the HP-UX kernel. (see Chapter 9, "Logical Volume Manager".)

run-level

A system state in which a specific set of processes are run. Run-levels are defined in `/etc/inittab`.

secondary loader

Code located in the Boot ROM which executes during system boot-up and loads the HP-UX system kernel. The Boot ROM loads and passes control to the secondary loader, which, in turn, loads and passes control to the file `/hp-ux` (or backup kernel, if `/SYSBCKUP` is specified).

secondary memory

The location (typically disks) where data is stored when not being used. By comparison, main memory (RAM) stores computer data required for program execution. Secondary memory (also called secondary data storage) is commonly termed swap. In the course of executing a program, data and instructions are swapped (that is, copied) to and from secondary memory.

secondary storage

The storage device(s), typically tape drives, used to back up and archive data stored on the system disks (primary storage). Secondary storage is also used to log transaction, interchange data and distribute software. Secondary storage devices always use removable media.

section

On Series 800 only, HP-UX allows you to divide a disk into separate pieces called sections. To the operating system, these pieces look like separate disk drives, each of which can have its own file system, swap area, or boot area. (See Chapter 8, "HFS File System".)

security policy

The set of laws, rules, and practices that regulate how an organization manages, protects, and distributes sensitive information.

select code

On a Series 300/400, the part of an address used for devices; a number determined by switch settings on the interface card. The select code determines the interface card's location in the processor address space. Each interface card is in turn connected to a peripheral. Multiple peripherals connected to the same interface card share the same select code. (See Chapter 11, "System Configuration".)

semaphore

A locking mechanism for ensuring that only one processor accesses critical kernel resources at a time on a multiprocessor (MP) system. (See Chapter 5, "Process Management".)

sensitive information

Information that, as determined by a competent authority, must be protected because its unauthorized disclosure, alteration, loss, or destruction will at least cause perceivable damage to someone or something.

session

Typically, a session is the same as a login shell and all the jobs spawned from the command line of that login shell. A session consists of one or more process groups. (See Chapter 5, "Process Management".)

session leader

Normally, the process that created the session (i.e., the login shell). (See Chapter 5, "Process Management".)

session lifetime

The period between when a session is created and the end of the process group lifetime of all its process groups. (See Chapter 5, "Process Management".)

set group ID bit

The middle bit of the most-significant octal digit in a file's protection mask. When set on a file that is an executable file, this bit causes the effective group ID of the user invoking the command to be set equal to the group ID of the file's group. (See Chapter 8, "HFS File System".)

set user ID bit

The most-significant bit of the most-significant octal digit in a file's protection mask. When set on a file that is an executable file, this bit causes the effective user ID of the user invoking the command to be set equal to the user ID of the file's owner. (See Chapter 8, "HFS File System".)

shared code

When an executable file is marked as having shared code, this means that the file's code can be shared among the processes that run it. That is, each process need not have its own copy of the code in its code segment; instead, there is one copy of the code, which all processes that run the program share. (See Chapter 7, "Memory Management".)

shared library

Object files linked to a program the first time the library is called. Subsequent calls to that library are mapped to the copy of the library already existing in memory. (See Chapter 7, "Memory Management".)

shared library segment

Each shared library segment typically has three `pregions`—text, initialized data, and `bss`. The shared library text can expand like other text segments, while the initialized data and `bss` cannot. (See Chapter 7, "Memory Management".)

shared memory segment

Shared memory segments are data segments that can be shared among processes. They are typically used when multiple processes must share data (for example, in a windowing system, all window processes must be able to update common data structures). Multiple shared memory segments can be attached by many processes using the *shmat*(2) system call. (see *shmat*(2) in the *HP-UX Reference* and Chapter 7, "Memory Management".)

shell

A program that interfaces between the user and the operating system. HP-supported shells are:

- `/bin/sh`
- `/bin/csh`
- `/bin/ksh`
- `/bin/rsh`
- `/bin/rksh`
- `/bin/pam`

shutdown

The process of taking the system from multi-user state to a single-user state, or taking the system from a running state to a state in which no processes are running.

shutdown command
: A shell script that has the primary function of terminating all currently running processes in an orderly and cautious manner. (See Chapter 3, "System Shutdown" and *shutdown*(1M) in the *HP-UX Reference*.)

slot
: The physical place in the card-cage of a computer where a card plugs in. Each slot has a number, which if the slot is used for modules, is multiplied by four to derive the module number (used in hardware addressing). (See Chapter 10, "System Architectures" and Chapter 11, "System Configuration".)

SMB
: System Main Bus, implemented on Models 850S, 855S, 860S, 865S, and 870S. Memory and processor boards are plugged into the SMB, which connects to the Mid-Bus for I/O via bus converter. (See Chapter 10, "System Architectures".)

soft limit
: In disk quotas, a soft limit (also termed quota) is the amount of file space allocated for a given user, and can be exceeded temporarily. (See Chapter 14, "System Accounting".)

source device
: The mass storage device from which HP-UX is installed. The source device must be a cartridge tape drive or flexible disk drive.

special file
: Often called a **device file**, this is a file associated with an I/O device. Special files are read and written just like ordinary files, but requests to read or write result in activation of the associated device. These files normally reside in the `/dev` directory. (See Chapter 11, "System Configuration".)

spinlock
: A kind of semaphore used to lock kernel resources very briefly. (See Chapter 5, "Process Management".)

stack segment

A memory segment reserved for use by a process's stack. On Series 300, it is attached near the top of the logical address space and grows "down" in memory. On Series 800, the stack and user data area share one space. (See Chapter 7, "Memory Management".)

standalone

A machine which is not part of an HP-UX cluster.

standard error

By default, all command error messages are displayed to a file known as standard error. Also, by default, standard error is displayed on the controlling terminal. (See Chapter 5, "Process Management".)

standard input

By default, all command input is taken from a file known as standard input. Also, by default, standard input is input (typed) at the keyboard of the controlling terminal. (See Chapter 5, "Process Management".)

standard output

By default, command output is displayed to a file known as standard output. Also, by default, standard output is displayed on the controlling terminal. (See Chapter 5, "Process Management".)

sticky bit

This is one of the bits on a file's protection mask. It can be set only by the system administrator via the `chmod` command. If set on an executable program, the program will continue to reside in the swap area until the system administrator unsets the sticky bit. This can improve the performance of programs that are often used since they can be loaded from the swap device, not form the file system. (See Chapter 8, "HFS File System".)

streaming

A design feature which allows a tape drive to move in long continuous motions rather than jerky motions. This is done by buffering data in such a way that it can be written smoothly. (See Chapter 12, "HP-UX Peripherals".)

subfile

Part of a CDF in an HP-UX cluster. The subfiles are under the hidden directory, and are named for one of the system's context attributes. (See *Managing Clusters of HP 9000 Systems*.)

subject

An active entity, generally in the form of a person, process, or device that causes information to flow among objects or changes the system state. Technically a process/domain pair.

superblock

A data structure containing global information about the file system such as file system size, disk information, and cylinder group parameters. The superblock is created at the same time as the file system and is replicated into each cylinder group. Also, HP-UX keeps a copy of the superblock in memory at all times. The `sync` command writes the superblock to disk. (See Chapter 8, "HFS File System" in this manual and *sync*(1M) in the *HP-UX Reference*.)

superuser

The root user who has special privileges. Same as the system administrator.

surface

In terms of optical disks, this is one of the disk sides—surface 1 or 2.

swap

Secondary memory (secondary data storage) is commonly termed swap. In the course of executing a program, data and instructions are swapped (that is, copied) to and from secondary memory. (See Chapter 7, "Memory Management".)

swapping

The means by which entire processes are moved for execution between RAM and mass storage (typically disk). (See Chapter 7, "Memory Management".)

swap space

Disk space which is reserved for use by the demand-paged virtual memory system. Processes are swapped to and from swap space. There are two kinds of swap space: primary swap space and dynamic swap space. (See Chapter 7, "Memory Management" and Chapter 8, "HFS File System".)

synchronization

When using LVM, synchronization keeps mirrored logical volumes consistent by ensuring that all copies contain the same data. (See Chapter 9, "Logical Volume Manager".)

system administrator run-level

A run-level of HP-UX when the system console provides the only communication mechanism between the system and its users. Init state s is the system administrator run-level. (See Chapter 6, "Run-Levels".)

system CDFs

System file (e.g., `/etc/rc`, `/etc/inittab`) that are also CDFs (context-dependent files). (See *Managing Clusters of HP 9000 Systems*.)

system console

A keyboard and display (or terminal) given a unique status by HP-UX and associated with the special device file `/dev/console`. All boot ROM error messages (messages sent prior to loading HP-UX), HP-UX system error messages, and certain system status messages are sent to the system console. Under certain conditions (for example, the single-user state), the system console provides the only mechanism for communicating with HP-UX. (See Chapter 2, "System Startup".)

system login script

A system-wide login script, run for every user of a particular shell when he or she logs in. System login scripts are:

- `/etc/profile` for Bourne and Korn shell users
- `/etc/csh.login` for C shell users.

system shutdown

The process of taking HP-UX from a running state to one in which no processes are running. See also **shutdown** and **reboot**. (See Chapter 3, "System Shutdown".)

system startup

The process of taking HP-UX from a stopped state (in which no processes are running) to a state in which it is ready to accept input from users. (See Chapter 2, "System Startup".)

tape density

A measure of how densely information is stored on a magnetic tape. **Bits per inch** is a common measure of tape density. (See Chapter 12, "HP-UX Peripherals".)

TCB

See **trusted computing base**.

TCSEC

Trusted Computer Systems Evaluation Criteria, also known as the "Orange Book". This is the book where evaluation criteria for trusted systems is documented.

text segment

See **code segment**.

thrashing

When the demand-paged virtual memory system spends an inordinate amount of time paging (swapping processes to and from swap space), so much so that system performance degrades. (See Chapter 7, "Memory Management".)

time-slice

The amount of time a process can run before the kernel checks to see if there is an equal-priority process ready to run.

track

One of several concentric circles on the surface of a disk upon which data is recorded (refer to Chapter 8, "HFS File System".)

translation lookaside buffer (TLB)

A page table that matches cache or indexes physical addresses (depending on design) for recently accessed virtual addresses. Once an address is translated, it can be used by the cache, which uses physical addresses. (See Chapter 7, "Memory Management".)

trap door
A hidden software or hardware mechanism that permits system protection mechanisms to be circumvented. It is activated in some non-apparent manner (e.g., special "random" key sequences).

Trojan horse
A computer program with an apparently or actually useful function that contains additional (hidden) functions that surreptitiously exploit the legitimate authorizations of the invoking process to the detriment of security—for example, making a "blind" copy of a sensitive file for the creator of the Trojan Horse.

trusted computer system
A system that employs sufficient hardware and software integrity measures to allow its use for processing simultaneously a range of sensitive or classified information.

trusted computing base
(TCB) The totality of protection mechanisms within a computer system—including hardware, firmware, and software—the combination of which is responsible for enforcing a security policy. A TCB consists of one or more components that together enforce a unified security policy over a product or system. The ability of a TCB to correctly enforce a security policy depends solely on the mechanisms within the TCB and on the correct input by system administrative personnel of parameters (e.g., a user's clearance) related to the security policy.

trusted process
A process that is restricted and is only run by a user with appropriate privilege. For HP-UX 6.5 a trusted process is a process with effective user ID of 0 (superuser).

trusted software
The software portion of a Trusted Computing Base.

u_area
A data structure containing process characteristics used by the kernel when a process is executing. When a process is swapped out, its u_area is also swapped out. (See Chapter 7, "Memory Management".)

unattended mode

The default method by which the boot ROM finds and loads an operating system. Occurs if the user does not intercede during system startup. (See Chapter 2, "System Startup".)

unit number

Part of an address used for devices; a number whose meaning is software- and device-dependent but which is often used to specify a particular disk drive in a device with a multi-drive controller. When referring to single-controller integrated disk/tape or disk/flexible disk drive, a unit is used to distinguish between disk and cartridge tape drives or hard disk and flexible disk drive. (See Chapter 11, "System Configuration".)

(The unit number also selects a single partition on the 913x series.)

use count

The demand-paged virtual memory system keeps track of the number of processes accessing shared code. This number is called the use count. (See Chapter 7, "Memory Management".)

user

Any person who interacts directly with a computer system.

user stack segment

Contains the run-time stack when a process is running in user mode. (See Chapter 7, "Memory Management".)

UUCP

Unix-to-Unix File Copy Package, mainly used for mail transport.

verification

In a trusted system, the process of comparing two levels of system specification for proper correspondence (e.g., security policy model with top-level specification, TLS with source code, or source code with object code). This process may or may not be automated.

virtual address space

The range of memory locations that the individual process can access, within the physical limits of swap space and main memory. (See Chapter 7, "Memory Management".)

vnode
A region is usually filled by a vnode, an interface for reading and writing pages of data between memory and disk. Vnodes are categorized by file type, such as `ufs`, `nfs`, `dux`, and `cdf`. (See Chapter 7, "Memory Management".)

volume group
In LVM, the combined disk space of one or more LVM disks (physical volumes) from which logical volumes space are allocated. (See Chapter 9, "Logical Volume Manager".)

volume number
Part of an address used for devices; a number whose meaning is software- and device-dependent but which is often used to specify a particular volume on a multi-volume disk drive. The volume number is also used to inform the device driver of special handling semantics (such as printer drivers skipping over perforations). (See Chapter 11, "System Configuration".)

word
A unit of information, typically consisting of four consecutive bytes (32 bits).

write protected
Indicates that a storage media is protected from being written to. This is useful in situations where you have a tape or diskette that you don't want written to and possibly destroyed. (See Chapter 12, "HP-UX Peripherals".)

write ring
A mechanism on a reel-to-reel tape that allows you to write-protect the tape.

Index

D

G

T

HEWLETT
PACKARD

Reorder No. or
Manual Part No.
B2355-90029

Manufacturing
Part No.
B2355-90029

B2355-90029