May 1995
Order Number: 312544-004

Paragon™ System

Administrator's Guide

Intel® Corporation

WARNING

Some of the circuitry inside this system operates at hazardous energy and electric shock voltage levels. To avoid the risk of personal injury due to contact with an energy hazard, or risk of electric shock, do not enter any portion of this system unless it is intended to be accessible without the use of a tool. The areas that are considered accessible are the outer enclosure and the area just inside the front door when all of the front panels are installed, and the front of the diagnostic station. There are no user serviceable areas inside the system. Refer any need for such access only to technical personnel that have been qualified by Intel Corporation.

CAUTION

This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at his own expense.

LIMITED RIGHTS

Preface

This manual explains how to administer a Paragon™ supercomputer. In broad terms, your task is to maintain two operating systems: Paragon operating system (which runs on the supercomputer) and SCO UNIX (which runs on the diagnostic station). You will need root privilege to do this. In this manual, "operating system" refers to the operating system that runs on the nodes of the Paragon(TM) supercomputer.

This manual is not an introduction to either the UNIX system or the Paragon system. Instead, the manual assumes that you already know how to administer a UNIX system and that you are familiar with the Paragon system. For general information on the Paragon system, see the *Paragon™ System User's Guide*. For general information on UNIX system administration, see your local technical bookstore.

Organization

Chapter 1	Introduces the system and your task as system administrator.
Chapter 2	Discusses how to start the system.
Chapter 3	Discusses system shutdown and recovery.
Chapter 4	Explains how to boot the system under various conditions.
Chapter 5	Explains how to customize the system to meet your site's needs.
Chapter 6	Explains how to add and remove users and groups.
Chapter 7	Discusses file backup and recovery.
Chapter 8	Explains how to configure your system's partitions of nodes.
Chapter 9	Discusses UFS and NFS File Systems.
Chapter 10	Discusses PFS File Systems.
Chapter 11	Discusses how to mange System I/O interfaces and devices.
Chapter 12	Explains how to configure paging trees.
Chapter 13	Discusses error logging.
Chapter 14	Discusses printer services.
Chapter 15	Discusses accounting for system usage.
Chapter 16	Explains how to detect and replace bad node boards.

Notational Conventions

This manual uses the following notational conventions:

Bold Identifies command names and switches, system call names, reserved words, and other items that must be used exactly as shown.

Italic Identifies variables, filenames, directories, processes, user names, and writer annotations in examples. Italic type style is also occasionally used to emphasize a word or phrase.

`Plain-Monospace`
Identifies computer output (prompts and messages), examples, and values of variables. Some examples contain annotations that describe specific parts of the example. These annotations (which are not part of the example code or session) appear in *italic* type style and flush with the right margin.

Bold-Italic-Monospace
Identifies user input (what you enter in response to some prompt).

`Bold-Monospace`
Identifies the names of keyboard keys (which are also enclosed in angle brackets). A dash indicates that the key preceding the dash is to be held down *while* the key following the dash is pressed. For example:

`<Break>` **`<s>`** **`<Ctrl-Alt-Del>`**

`[ ]` (Brackets) Surround optional items.

`...` (Ellipsis dots) Indicate that the preceding item may be repeated.

`|` (Bar) Separates two or more items of which you may select only one.

`{ }` (Braces) Surround two or more items of which you must select one.

Applicable Documents

For more information, refer to the *Paragon™ System Technical Documentation Guide*.

Comments and Assistance

Intel Scalable Systems Division is eager to hear of your experiences with our products. Please call us if you need assistance, have questions, or otherwise want to comment on your Paragon system.

U.S.A./Canada Intel Corporation
Phone: 800-421-2823
Internet: support@ssd.intel.com

Intel Corporation Italia s.p.a.
Milanofiori Palazzo
20090 Assago
Milano
Italy
1678 77203 (toll free)

France Intel Corporation
1 Rue Edison-BP303
78054 St. Quentin-en-Yvelines Cedex
France
0590 8602 (toll free)

Intel Japan K.K.
Scalable Systems Division
5-6 Tokodai, Tsukuba City
Ibaraki-Ken 300-26
Japan
0298-47-8904

United Kingdom Intel Corporation (UK) Ltd.
Scalable Systems Division
Pipers Way
Swindon SN3 IRJ
England
0800 212665 (toll free)
(44) 793 491056
(44) 793 431062
(44) 793 480874
(44) 793 495108

Germany Intel Semiconductor GmbH
Dornacher Strasse 1
85622 Feldkirchen bei Muenchen
Germany
0130 813741 (toll free)

World Headquarters
Intel Corporation
Scalable Systems Division
15201 N.W. Greenbrier Parkway
Beaverton, Oregon 97006
U.S.A.
(503) 677-7600 (Monday through Friday, 8 AM to 5 PM Pacific Time)
Fax: (503) 677-9147

If you have comments about our manuals, please fill out and mail the enclosed Comment Card. You can also send your comments electronically to the following address:

techpubs@ssd.intel.com
(Internet)

Table of Contents

Chapter 1
Introduction

Chapter 2
Starting the System

Chapter 3
System Shutdown and Recovery

Chapter 4
Booting the System

Chapter 5
Customizing the System

Chapter 6
Adding and Removing Users and Groups

Chapter 7
Backing Up and Restoring Files

Chapter 8
Configuring Partitions

Chapter 9
Managing UFS and NFS File Systems

Chapter 10
Managing PFS File Systems

Chapter 11
Managing I/O Interfaces and Devices

Chapter 12
Paging Trees

Chapter 13
Error Logging

Chapter 14
Printer Services Management

Chapter 15
Accounting

List of Illustrations

List of Tables

Introduction 1

This chapter introduces the Paragon™ operating system, the hardware on which it runs, and the tasks which you will perform as system administrator. Refer to the *Paragon™ System User's Guide* for more detailed information about the operating system and its use in developing parallel applications. The remaining chapters of this manual provide detailed descriptions of system administration tasks.

Hardware Overview

The operating system system runs on several models of Intel supercomputers. These systems all have a large number of *nodes*. The nodes are connected by a high-speed *node interconnect network* and are grouped into *partitions* that support different types of activities.

Nodes

Each node is essentially a separate computer, with one or more i860® processors and 16M bytes or more of memory. Nodes can run distinct programs and have distinct memory spaces. They can team up to work on the same problem and exchange data by passing messages. Each node is used in one or more of the following ways:

Compute node — A node that is used for compute-intensive, parallel applications.

Service node — A node that is used for interactive processes such as shells and editors. (As system administrator, you can designate whether a node will be a compute node or a service node, but there are no physical differences between the two.)

I/O node — A node that is equipped with a SCSI interface, an Ethernet interface, a HIPPI interface, or some other I/O connection. I/O nodes manage the system's disk and tape drives, network connections, and other I/O facilities. I/O nodes communicate with the other nodes over the node interconnect network. However, this access is transparent: processes on nodes without I/O hardware access the I/O facilities using standard system calls, just as though they were directly connected. Nodes

with I/O interfaces are otherwise identical to nodes without I/O interfaces and can run user processes. An I/O node can also function as a compute node or service node, or can be dedicated to I/O tasks. I/O nodes are sometimes called *MIO* (for *Multipurpose I/O*) nodes.

Boot node — A special I/O node from which the system boots. Following a successful boot, this node functions as both a service node and an I/O node.

A Paragon supercomputer can have up to 2000 nodes. Each node can run more than one process at the same time; these processes can belong to the same or different applications.

There are three kinds of physical nodes:

MP node — A three-processor node that runs with two processors as general processors and one processor as message co-processor.

GP node — A two-processor node that runs with one processor as a general processor and one processor as a message co-processor.

MP-as-GP node — A three-processor node that runs with one processor as a general processor, one processor as a message co-processor, and one processor turned off.

An MP system is a Paragon supercomputer that is configured with MP nodes only. A GP system is a Paragon supercomputer that is configured with either GP nodes or MP-as-GP nodes only. In a GP system, MP nodes are automatically configured as MP-as-GP nodes. For a physical description of the nodes as well as installation instructions, refer to *Paragon™ System Hardware Installation Manual*.

Node Partitions

The nodes in a Paragon system are partitioned (grouped) in the following ways:

Root partition — All the nodes in the system.

Compute partition — A collection of compute nodes. The compute partition is used to run applications and is a subset of the root partition.

Service partition — A collection of service nodes. The service partition is used to run normal housekeeping tasks and is a subset of the root partition.

I/O partition (optional) — A collection of I/O nodes. The I/O partition is a subset of the root partition. I/O nodes can also be placed in the service and/or compute partition.

As system administrator, you have some degree of control over all the nodes in the system. See Chapter 8 for more information on partitions.

Node Interconnect Network

The nodes are connected by a high-speed *node interconnect network*. Each node interfaces to this network through special hardware that monitors the network and extracts only those messages addressed to its attached node. Messages addressed to other nodes are passed on without interrupting the node processor. For most purposes, you can think of each node as being fully connected to all the other nodes; the time required to send a message to the most distant node in the system is only slightly greater than the time to send a message to a neighbor node.

Hardware Platforms

Currently there are two Paragon hardware platforms: Paragon XP/S System and Paragon XP/E System. These two platforms run identical software and differ only in their hardware. Unless otherwise noted, all sections of this manual refer to both platforms.

The next two sections discuss the hardware components of the Paragon XP/S System and the Paragon XP/E System, respectively. Read the section that applies to your system.

Paragon™ XP/S System Hardware

A Paragon XP/S System system consists of the following basic hardware components:

- Two or more cabinets. The cabinets are numbered sequentially (0, 1, 2, etc.) from right to left (as viewed from the front).
- An internal diagnostic station (contained in cabinet 0).
- An Ethernet link connecting users' workstations to the system.

Figure 1-1 shows a two-cabinet Paragon XP/S System system (the diagnostic station is hidden behind the door of cabinet 0).

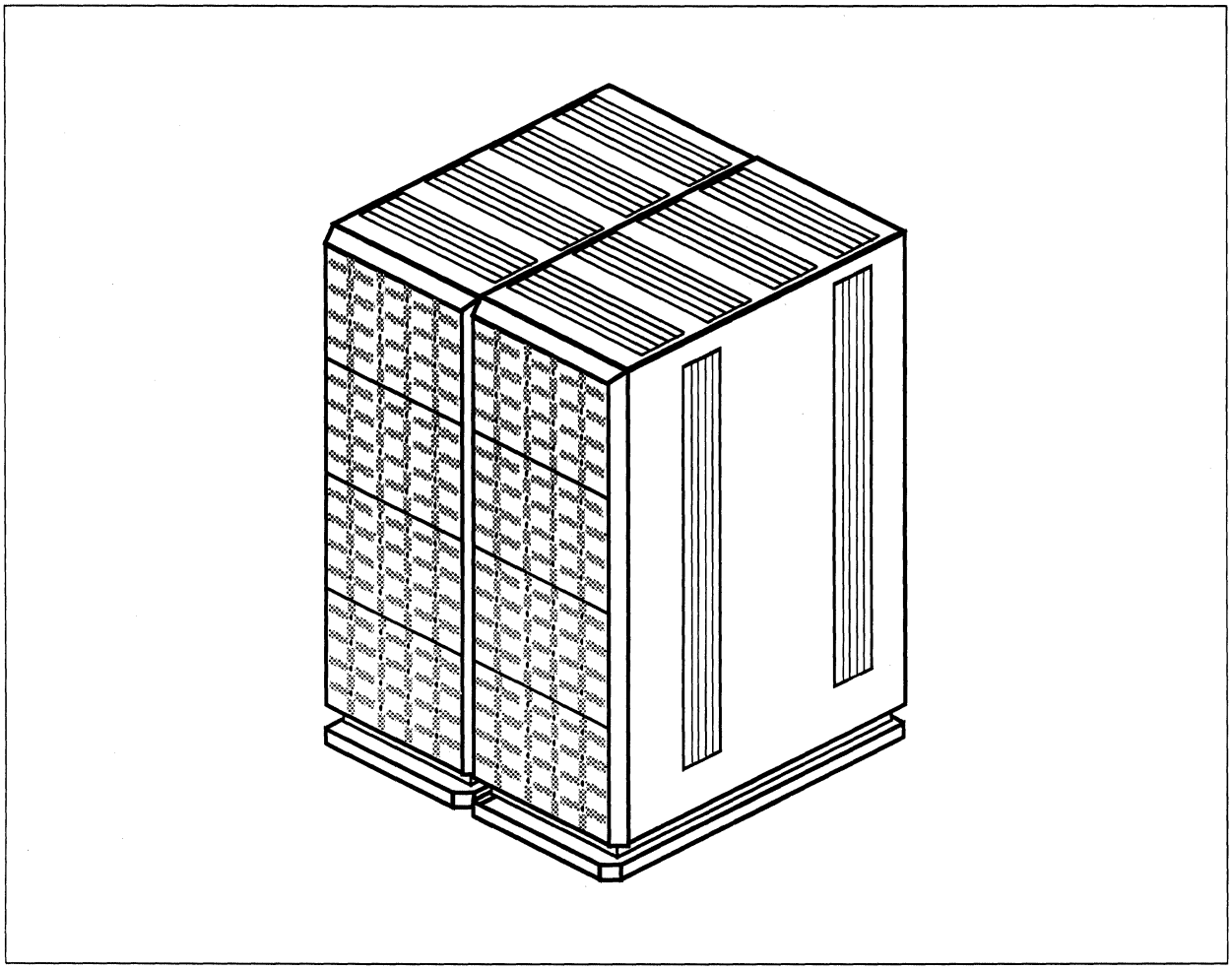


Figure 1-1. Hardware Components of a Paragon™ XP/S System System

Paragon™ XP/S System Cabinets

A Paragon XP/S System system always contains at least two cabinets, numbered 0 and 1. Cabinet 0 houses the diagnostic station. The system can also contain additional cabinets, numbered 2, 3, etc. The front door of each cabinet contains an LED panel.

The cabinets house the system's computational hardware, mesh I/O hardware, and mass-storage peripherals. Each cabinet contains an industry-standard 19-inch rack, DC power module(s), door LED circuitry, AC power circuitry, cooling modules, up to four cardcages, and up to three swing-out peripheral modules.

Each cardcage contains seventeen slots, sixteen for node boards and one for a performance monitor board. The performance monitor board is optional. If it is not present, the slot is empty. The same slots can be used for any type of node (compute, service, I/O, or boot), although the boot node should be located in the upper-right corner of the system (as viewed from the front) and the service and I/O nodes are typically located on the right and/or left edge of the system.

Each peripheral module holds up to three RAID-3 subsystems. RAID stands for Redundant Array of Inexpensive Disks; each RAID-3 subsystem contains five 3.5-inch disk drives for a total of 4G bytes or more of formatted storage. These drives appear as one device to the processor. Data is striped across the drives. The RAID-3 subsystem maintains parity on the data, so if one drive fails the data remains intact, and operation can continue. Each peripheral module can also be configured to hold 4-mm DAT tape drives. These drives transfer data at up to 180K bytes/sec and hold up to 2G bytes of data per tape.

Figure 1-2 on page 1-6 shows what cabinet 0 looks like when the front-door LED panel is open. The figure shows the diagnostic station and six RAID-3 subsystems in two peripheral modules. Compute and I/O node boards reside behind the RAID-3 modules. The diagnostic station and peripheral modules are hinged on the right to swing open left-to-right.

Paragon™ XP/S System Diagnostic Station

The Paragon XP/S System diagnostic station is an Intel486™-based computer that is used to administer the Paragon XP/S System system. The diagnostic station has 32M bytes of memory, a 540M-byte SCSI hard disk drive, a 0.25-inch cartridge tape drive, a 3.5-inch diskette drive, a 101-key keyboard, a 14-inch flat-panel display, and a trackball.

The diagnostic station runs the SCO UNIX operating system. The user interface, which is called the Diagnostic and Test Executive (DTX), runs under the SCO Open Desktop. DTX allows you to test the system and obtain information about system configuration and status. You can access the diagnostic station remotely through its Ethernet connection.

Paragon™ XP/E System Hardware

A Paragon XP/E System system consists of the following basic hardware components:

- One Paragon XP/E System cabinet.
- An external diagnostic station.
- An Ethernet link connecting users' workstations to the cabinet.

Figure 1-3 shows a Paragon XP/E System system (the table on which the diagnostic station sits is not included).

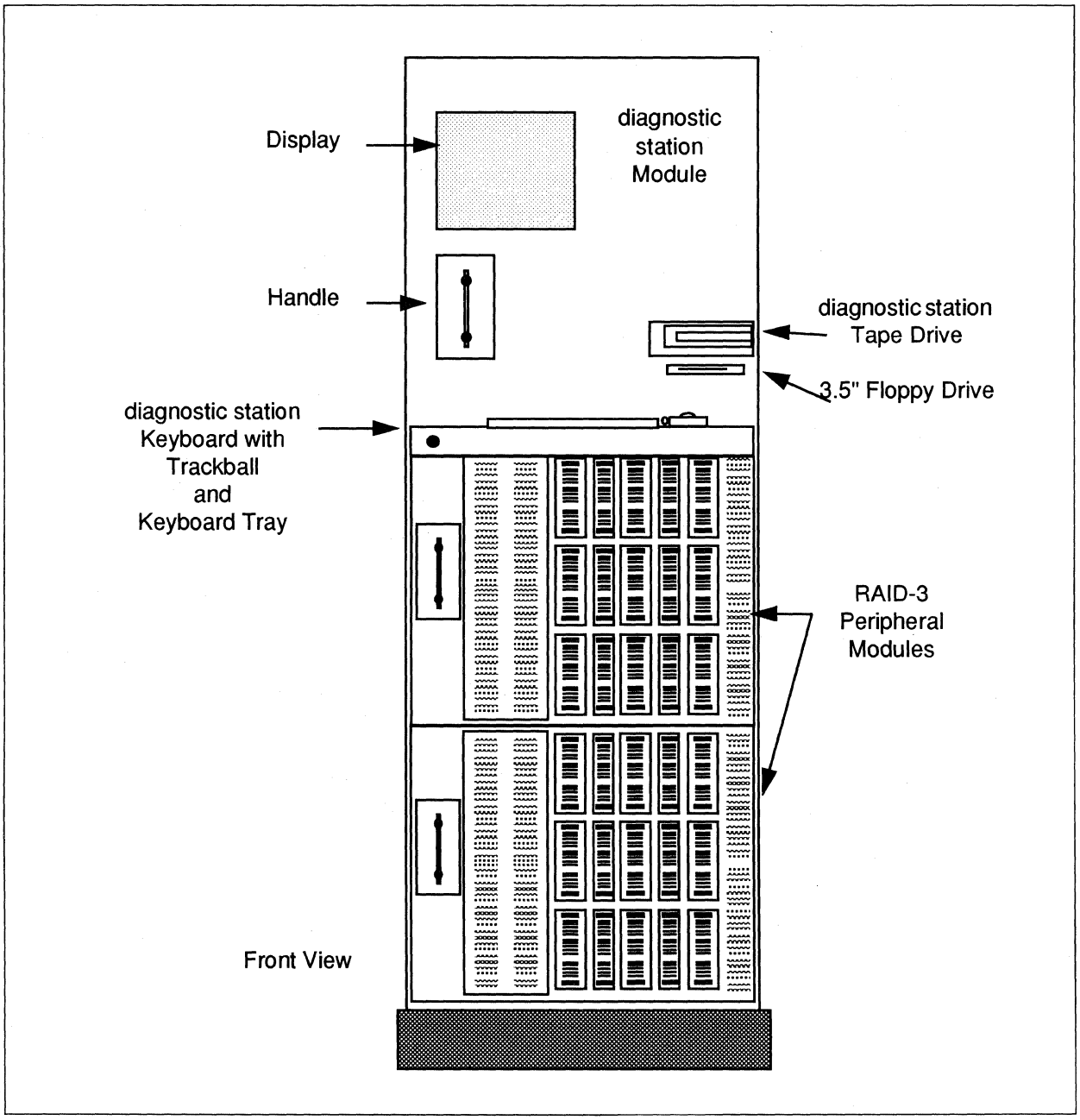


Figure 1-2. Paragon™ XP/S System Cabinet 0 Components

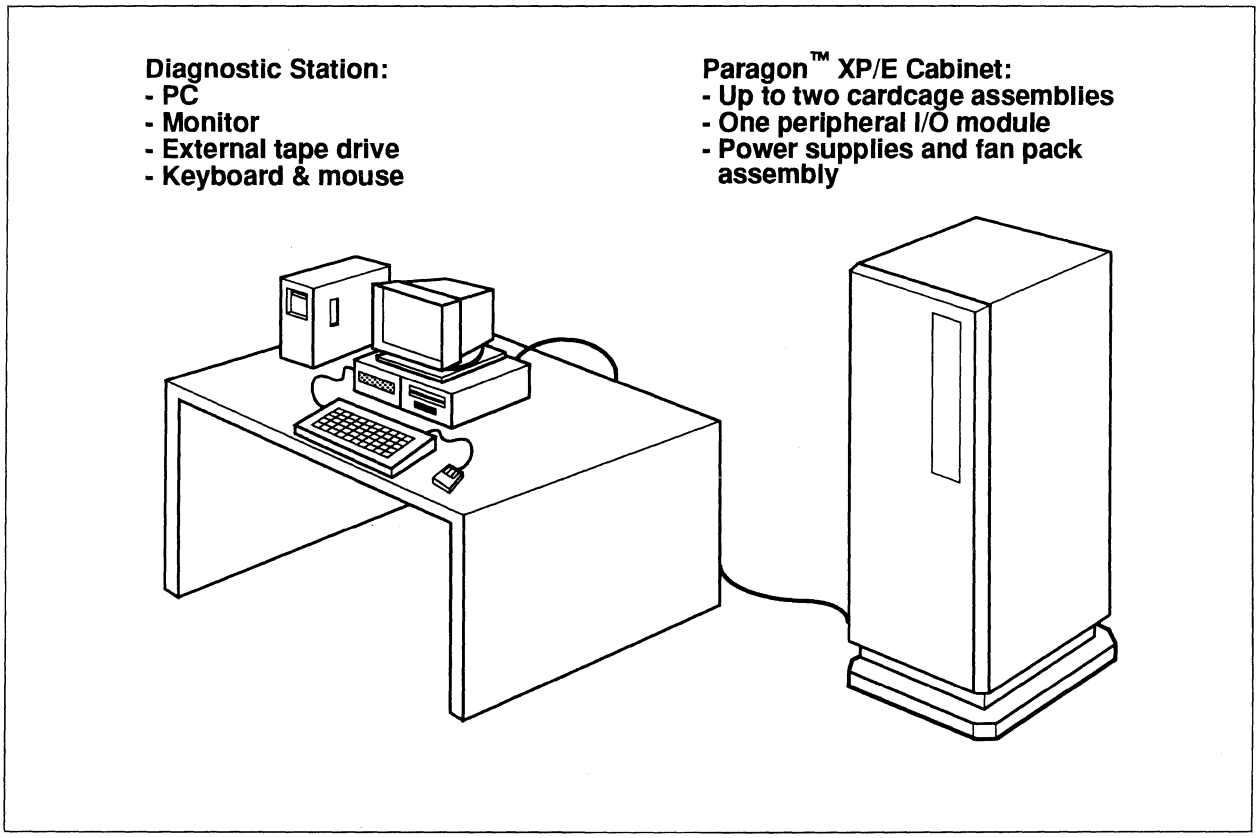


Figure 1-3. Hardware Components of a Paragon™ XP/E System System

Paragon™ XP/E System Cabinet

A Paragon XP/E System system includes a single cabinet. A green power on/off indicator is located in the upper right corner of the cabinet's front door. The cabinet itself contains a peripheral I/O module, up to two cardcages, a fan assembly, up to three power supplies, an I/O panel, and an AC service area.

The base peripheral module configuration includes a tape drive and two individual disk drives. The tape drive is a 4-mm DAT drive that can transfer data at 180K bytes/sec. Its tape cartridge can hold up to 2G bytes. The disk drives are standard 3.5-inch drives. The peripheral module may also contain additional tape or disk drives and up to two RAID-3 subsystems, like those of the Paragon XP/S System.

The base cardcage configuration includes one or two cardcages, each with up to 16 nodes and a performance monitor board slot, like those of the Paragon XP/S System. These cardcages are populated with four to twenty-eight compute nodes (depending on which model you have), one boot node (an I/O node that is used to boot the system), and one service node (a compute or I/O node that belongs to the service partition). The cardcages may also contain additional I/O nodes.

Power supplies provide ± 5 volts and ± 12 volts to the node boards, disk drives, tape drives, and fans. AC power input to these supplies is single-phase, 220 volts.

Figure 1-4 on page 1-9 shows the base configuration for a Paragon XP/E System 16N system. This system contains a 4-mm DAT drive, two individual disk drives, sixteen compute nodes, one boot node, and one service node. The shaded cardcage slots indicate vacant performance monitor board slots.

Paragon™ XP/E System Diagnostic Station

The Paragon XP/E System diagnostic station is similar to the diagnostic station of the Paragon XP/S System, except that it is external to the system (instead of being housed inside the cabinet) and has its own power supply. It also uses a CRT display and a mouse instead of a flat-screen display and a trackball. Apart from these differences, it has the same specifications and runs the same software as the Paragon XP/S System diagnostic station.

Node Numbering

There are two node numbering schemes in the Paragon system: *CBS* (Cabinet, Backplane, Slot) numbering, and *root* node numbering (also called *OS* numbering).

You should become familiar with both schemes, because you will encounter them both. For example, the *SYSCONFIG.TXT* file uses CBS numbering because it is primarily a hardware configuration file, and the *MAGIC.MASTER* file uses root node numbering because it is primarily a software configuration file.

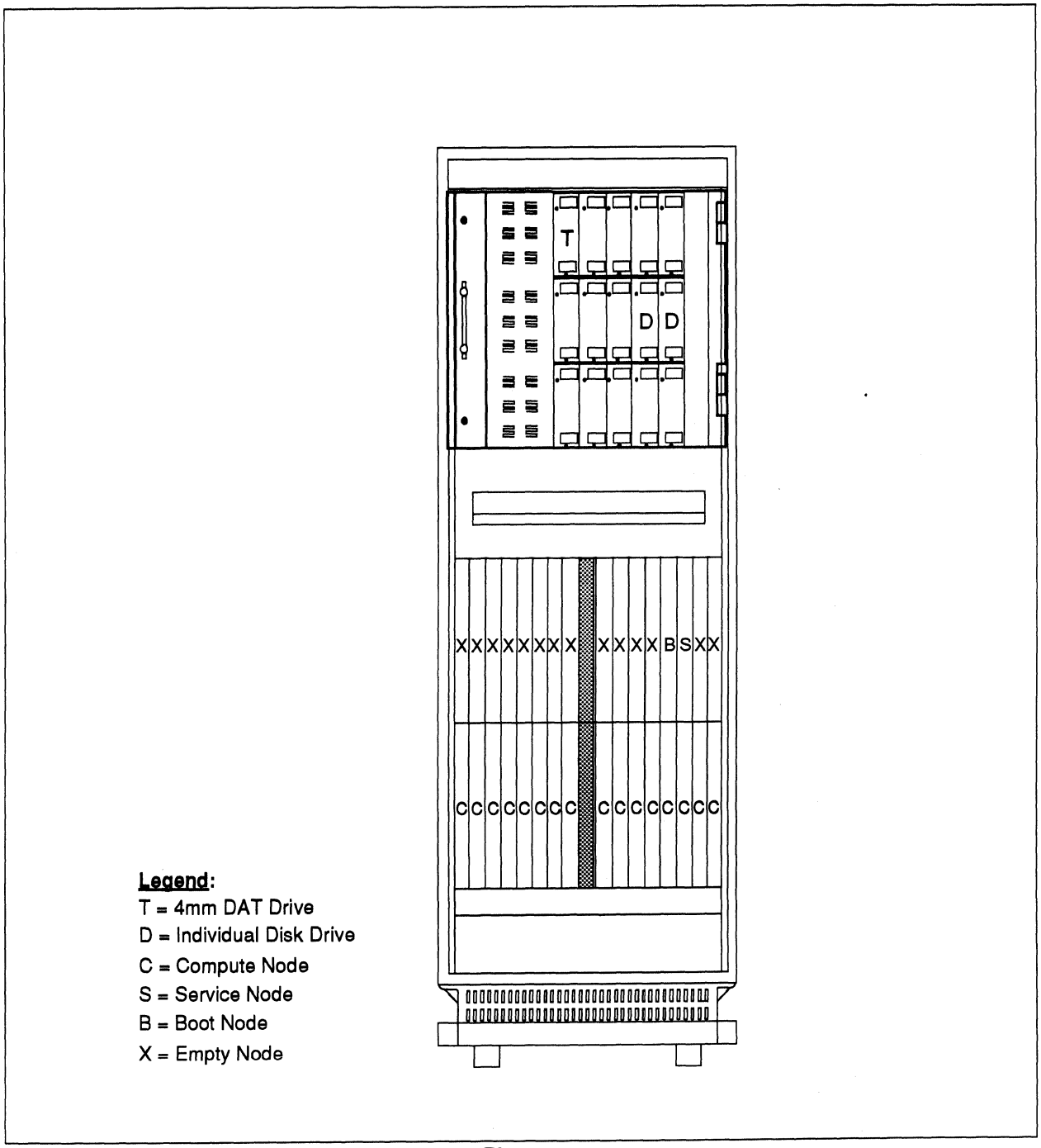


Figure 1-4. Paragon™ XP/E System 16N System

CBS Numbering

A CBS node number has the form *ccBss*, where *cc* is the two-digit number of a cabinet in the system, *B* is a single letter identifying the backplane within that cabinet, and *ss* is the two-digit slot number identifying the slot within that backplane.

The following example shows the first few lines of a sample *SYSCONFIG.TXT* file, which illustrates how the CBS numbers are used.

```
CABINET 0
PC xx OK
LED xx
BP D xx
S 0 GPNODE xx 16 MRC xx
S 1 GPNODE xx 16 MIO xx RAID3 xx MRC xx
S 2 GPNODE xx 16 MIO xx RAID3 xx MRC xx
S 3 GPNODE xx 16 MIO xx RAID3 xx DAT xx ENET xx MRC xx
S 4 GPNODE xx 16 MRC xx
```

In this example, the **CABINET** line indicates cabinet 0, the **BP** line indicates backplane D in that cabinet, and the five **S** lines indicate slots 0 through 4 in that backplane. Thus, the last line of the example describes the node whose CBS number is 00D04. For more information about CBS numbering, refer to the manpages for the **cbs** command.

Cabinets

Cabinet numbering begins with Cabinet 0, which is the right-most cabinet as viewed from the front, and proceeds to the left (as viewed from the front).

Backplanes

There are up to four backplanes in each cabinet. The backplanes are identified alphabetically, starting with A at the bottom. Each backplane consists of up to 16 nodes, which are arranged linearly in slots.

Slots

When the nodes are viewed from the front, the slot numbering starts at the right of the card cage and moves left. (The performance monitor slot in the middle of each card cage is not counted.) To find the LED associated with a slot, you begin counting at the lower-right corner of the backplane's block of LEDs, count up the right column, and then over to the next column to the left, and then back down in a serpentine fashion; that is, slot 0 is the lower-right LED, and slot 15 is the lower-left LED.

Figure 1-5 shows how the slots in a single backplane map to the LEDs in the cabinet door.

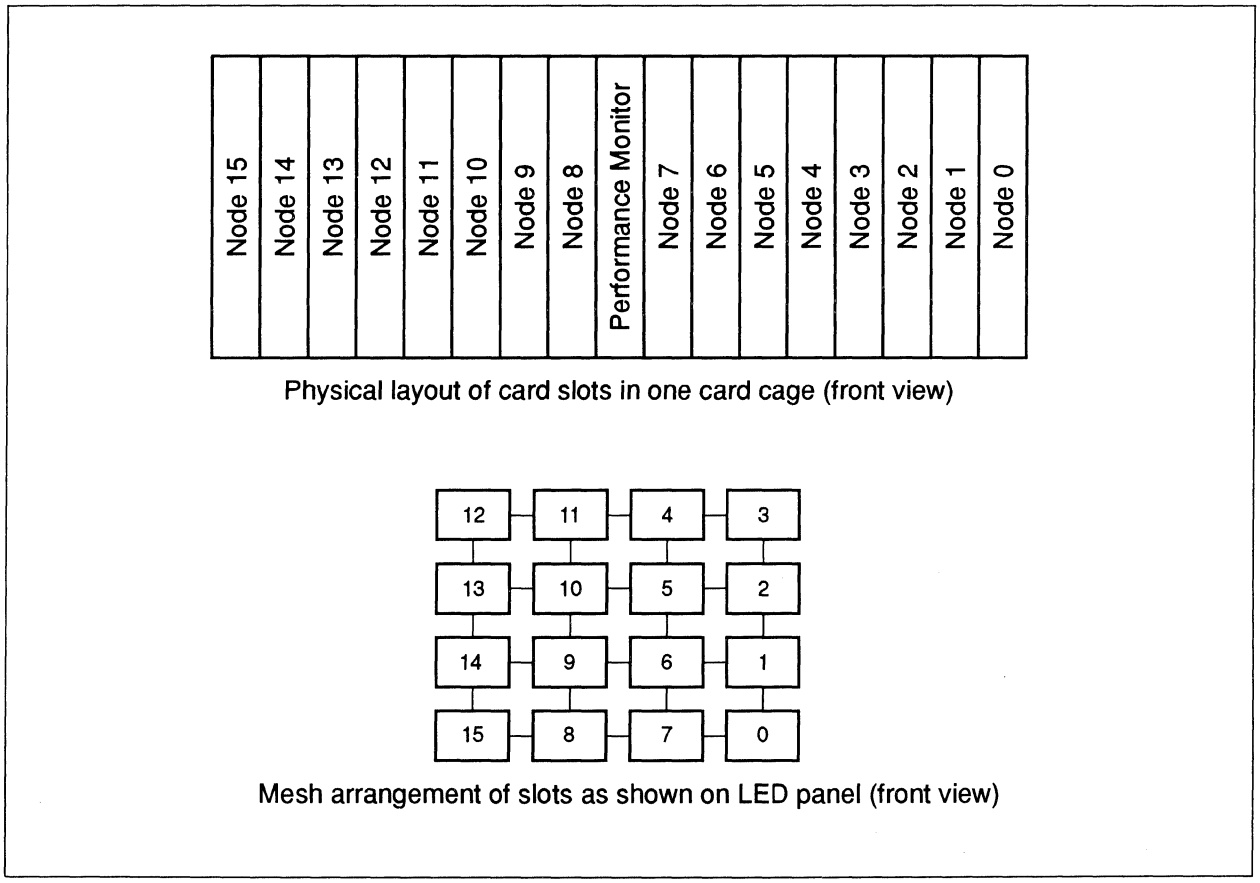


Figure 1-5. CBS Node Numbering

Root Node Numbering

Root node numbering starts at the top left of the left-most cabinet, and spans all of the cabinets in the Paragon system. When you reach the end of the right-most cabinet, return to the left-most cabinet, drop down a row, and continue counting. These numbers are the node numbers used within the *root* partition (see Chapter 8 for information on the *root* partition). *Root* node numbering is also called *OS node numbering*, because it is used by the operating system.

Figure 1-6 shows the *root* node numbers and the CBS numbers for all the nodes in a two-cabinet system. Each square in the figure represents an LED; the top (bold) number in each square is the *root* node number for the corresponding node, and the bottom (plain) number is the CBS number for that node. For example, the node with *root* node number 21 has CBS number 0D09.

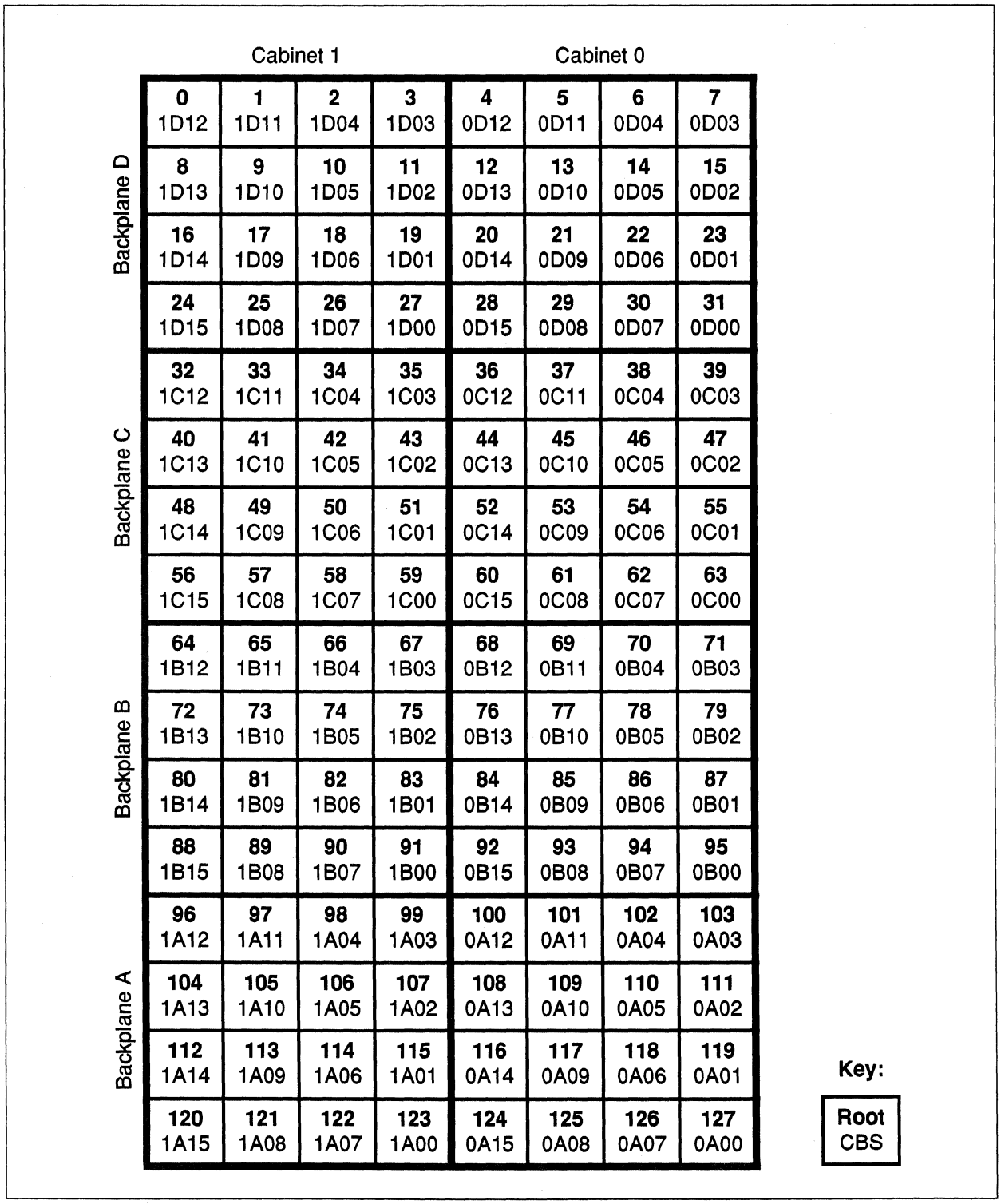


Figure 1-6. CBS and Root Node Numbering in a Two-Cabinet System

Ethernet and Remote Workstations

Remote workstations are linked to a Paragon system by an Ethernet connection. Remote workstations are intended primarily for ordinary users, but in many cases you can also use them to perform diagnostic operations. The diagnostic station, however, is intended solely for system administration and should not be used as a remote workstation.

System Software

Paragon system software consists of:

- Two operating systems (one for the supercomputer and one for the diagnostic station).
- Paragon system extensions.
- Utilities.

Node Operating System

Each node of the Paragon system runs the operating system operating system. For more information about the operating system operating system, refer to the *Paragon™ System User's Guide* and the *OSF/1 System and Network Administrator's Reference*.

Diagnostic Station Operating System

The diagnostic station runs the SCO Open Desktop V1.1 operating system, which is based on System V R5.32. Refer to the *Open Desktop Administrator's Guide* for more information.

Paragon™ System Extensions

The Paragon system extensions include application libraries, message-passing facilities, high-performance I/O facilities, and system commands. The application libraries provide Paragon system calls that are available to applications running on the nodes. The system commands include both commands that are available to regular users and commands that are only available to the system administrator (logged in as *root*). See the *Paragon™ System Commands Reference Manual* for more information about both types of commands.

Utilities

The utilities include batch queueing, accounting, and performance analysis utilities as well as a full suite of application development tools (debugger, performance analysis tool, etc.).

System Administration Tasks

The system administrator is the person with the most responsibility for the system.

As system administrator, you are the "keeper of the *root* password." Because your actions can affect the entire system, you should not reveal the *root* password to others unless it is absolutely necessary. And when those others no longer need *root* access, you should change the *root* password. If several people are doing root-level activity and not communicating with each other, a system's configuration can easily enter an unknown state. You should know the system's configuration at all times.

You should also treat *root* activity with care. Use *root* privilege only when it is really necessary to do so. If you can perform a task without *root* privilege, you should do so. This minimizes the consequences of any errors you may make.

You will be actively involved in system operations at the following times:

- *At system setup*, or whenever you need to shut the system down and then reboot it (when adding new hardware or software, for example). Chapter 3 and Chapter 4 describe these tasks.
- *On a routine basis*. This includes tasks that are performed more often than setup tasks. Examples include customizing the system environment, managing printer services, and managing file systems. Chapter 5 through Chapter 14 describes these tasks.
- *When tracking system usage*. The system administrator is typically responsible for scheduling and accounting activities that monitor the usage of system resources. Chapter 15 describes these tasks.
- *When system problems need to be diagnosed and solved*. The system administrator is the first person contacted when the system fails to operate properly for any reason. Refer to the *Paragon™ System Diagnostic Reference Manual* for more information about diagnostics. Chapter 16 explains how to detect and replace bad node boards.

The following sections briefly summarize the setup, routine, and scheduling and accounting tasks.

Setup Tasks

Setup tasks are performed occasionally (at system setup, for example). Setup tasks include:

- Installing, reinstalling, and updating system software.
- Recording the hardware configuration of the system (done when the system is first set up and then only when the hardware changes).
- Creating the service and compute node partitions.
- Configuring PFS file systems.
- Establishing network addresses for TCP/IP nodes (usually done only once).
- Determining ULA addresses for HIPPI boards (usually done only once).
- Powering up and powering down the system.

Routine Tasks

Routine tasks are performed on a regular, ongoing basis to ensure that the system runs smoothly. Routine tasks include:

- Changing the root password.
- Miscellaneous hardware maintenance tasks.
- Maintaining the diagnostic station (backing up and restoring its system disk).
- Managing the RAID facilities (XP/S and some XP/E systems).
- Monitoring system metrics such as the mesh-usage network bandwidth.
- Certain Remote Host operations, such as maintaining system security for remote host operations.

Scheduling and Accounting Tasks

Scheduling and accounting tasks are performed on a periodic basis to track system usage. These tasks include:

- Logging system operations.
- Supporting external user accounting functions.
- Performing batch or open partition scheduling tasks.
- Generating accounting reports.

Starting the System 2

This chapter describes the factors that influence the procedures and commands you use to start or reboot the system:

- The boot operation.
- The different start-up states and the corresponding boot preparation.
- Manipulating the different run levels.
- Setting the system clock.

NOTE

Refer to the system Release Notes for information about installing the system and performing the initial boot. The information in this chapter assumes that you are rebooting an installed operating system.

NOTE

The following subsections describe the system conditions that you must understand *before* booting your system. To understand the many facets of booting the Paragon system, refer to Chapter 4.

Terms and Concepts

The following terms and concepts are crucial to understanding the boot process:

boot node — The first node on which the Paragon operating system is booted. An application-level program running on the boot node then initiates booting of the remaining nodes on the mesh. Any node may serve as the boot node as long as the following criteria are met:

- The boot node must be an MIO card in a corner of the mesh.
- An Ethernet connection must exist between the diagnostic station and the boot node.
- The boot node must have access to mass storage (a hard disk) that contains the *root* file system.

compute nodes — The nodes on the mesh that are booted from the boot node.

bootmagic strings — A set of descriptors of the form <*name*>=<*value*> that are required by the booting process.

bootmagic — The *bootmagic* configuration file resides on the diagnostic station's file system. The *bootmagic* file is dynamically prepared at boot time by the *boot preprocessor* to reflect the current configuration of the system (see the boot preprocessor manual page, **bootpp**). Information from *bootmagic* configures the microkernel, the operating system servers, and application-level partitions. The *bootmagic* file is downloaded to the microkernel, which makes it available to the operating system server and user-level utilities, such as the *mesh booter*. The user should never modify the *bootmagic* configuration file.

boot preprocessor — The boot preprocessor, **bootpp**, executes from the diagnostic station and prepares information in the *bootmagic* file at boot time, using the contents of the *MAGIC.MASTER* and *SYSCONFIG.TXT* files. The **bootpp** boot preprocessor uses a master copy of *bootmagic* and a hardware configuration file (dynamically prepared by the Diagnostics subsystem) to generate a snapshot of the current mesh configuration and other information needed during the boot process.

mesh booter — The **bootmesh** utility runs during the booting sequence on the boot node only. Using configuration information from *bootmagic*, the mesh booter finds files that contain the microkernel and the operating system server. These files are then downloaded to all of the other nodes on the mesh. When the loading of the mesh completes, the **bootmesh** utility initiates microkernel execution on each mesh node and then exits.

root partition — A rectangle of nodes, numbered row-wise, left-to-right, and top-to-bottom. The *root* partition includes all of the slots in the system, whether or not the slots are populated.

Powering Up the Paragon™ Hardware

The Paragon hardware must be powered up before you can boot the operating system. Use the following procedure if the system is not already powered up.

SAFETY WARNING

To minimize the risk of electric shock and energy hazards, remove any watches, rings, bracelets, or necklaces that may be conductive before opening the rear door or any front panels. Also, to minimize accidental contact, immobilize one arm and hand by placing the hand in a pocket or behind your back before entering the service access areas behind the rear door and front panels. Assure that each movement is deliberate and do not contact any parts until a correct positive identification has been made. If there is any question in your mind about how to proceed in these locations, stop and refer to your SSD Customer Service representative before proceeding.

1. Locate the circuit breakers at the bottom on the back of the each cabinet.

   If the circuit breakers are already up, there's no need to cycle them. Once the breakers are on, they should not be turned off.

   If the breakers are down (off), flip them up starting with cabinet 0 and proceeding in numerical order through the rest of the cabinets. Cabinet 0 is the cabinet containing the diagnostic station, and is the leftmost cabinet when viewed from the back or the rightmost when viewed from the front.

2. Open the back door of any cabinet. Notice the green power control board with red LEDs. This board has three white buttons on its upper-right corner.

   A. Press the bottom button to shut down power to the nodes.

   B. Wait about ten seconds for the disks to spin down and then press the top button to power up the nodes.

3. Power on the diagnostic station.

   A. The diagnostic station is located in cabinet 0, which is the rightmost cabinet when facing the cabinets from the front. Open the front door of this cabinet and swing out the diagnostic station.

   B. Turn on the power switch located on the back of the diagnostic station.

Preparing to Boot the Installed System

As the system administrator, you set up or encounter various pre-boot or post-shutdown states. This section describes the potential variety of states and recommends the steps that you can take before initiating the actual reboot. The scenarios described here are not extensive but they offer a sketch of the circumstances that you might encounter either routinely or periodically. The potential states described here include:

- A powered-down system.
- A powered up, halted system.
- A powered up system in single-user mode.
- A system that has crashed.

NOTE

The following subsections describe the system conditions that you must understand *before* booting your system. To understand the many facets of booting the Paragon system, refer to Chapter 4.

Preparing to Boot a Powered-Down System

In a powered-down system the hardware is offline and the processors in the mesh are stopped. Booting from this state is sometimes called a cold boot and may only be accomplished through the diagnostic station. Cold boots are common for administrators who power down the hardware periodically for routine maintenance or to configure new devices. If you are preparing to reboot your system from a powered down state, follow these steps:

1. Confirm that the hardware and all peripheral devices are connected. This may seem obvious, but it's surprising how easy it is to overlook connections when you are eager to restart and test a system with new devices.

2. Power up the hardware and peripheral devices, as described under "Powering Up the Paragon™ Hardware" on page 2-3. Again, this may seem obvious. Nevertheless, it is not unusual to forget to power up a device that you powered down earlier. If your installation is large, and if the interval between the power down and the reboot is long, the odds are good that you could forget.

3. Confirm that the hardware completed its restart and diagnostic operations. Most hardware provides a diagnostic check as a routine part of its start-up operation. Refer to the *Paragon™ System Diagnostic Reference Manual* for information on how to run diagnostics.

4. Wait for the diagnostic station prompt. The default prompt is *DS* and a hash mark (`DS#`).

5. Follow the procedure under "Preparing to Boot a Powered Up, Halted System" on page 2-5 to complete the boot process.

Preparing to Boot a Powered Up, Halted System

When your machine is powered up and enabled but the processors are halted, the system is in console mode. For example, if you have just powered the system up, or if you have shut down the processors with the **shutdown -h** command or have halted the processors with the **halt** command, your system displays the console prompt (`DS#`). When the system displays the console prompt, follow these steps from the diagnostic station:

1. Decide which start-up mode you want to initiate.

   - If you have tasks you need to accomplish and want the system to restrict access to all users but *root*, plan on booting to single-user mode.

   - If you do not require solo access, and you want the system to initialize full functionality, plan on booting to a multi-user mode.

2. Enter the boot command that corresponds to the desired start-up mode, as described under "Identifying the System Run Levels" on page 2-7.

Preparing to Boot a Powered Up System From Single-User Mode

When your machine is powered up and enabled with only a single node running, the system is said to be in single-user mode. When the Paragon operating system displays the *root* prompt (#), follow these steps from a system console that is connected to the mesh through Ethernet:

1. Decide whether you should continue in single-user mode or initiate a boot to a multi-user mode.

   - If you have additional tasks that you need to perform and want the system to deny access to all users but *root*, plan on continuing in single-user mode.

- If you do not require solo access, or if you have completed your tasks and you want the system to initialize full functionality, plan on booting to a multi-user mode.

2. When you are ready to boot to a multi-user mode, select the required command. "Identifying the System Run Levels" on page 2-7 describes the commands and procedures for booting to a multi-user mode.

Preparing to Boot A System that Crashed

A system that crashed presents a variety of unique requirements. The nature of the crash determines whether the crash is a catastrophe, a near catastrophe, or a serious problem that you must resolve as quickly as possible.

NOTE

System administrators can use the **autoddb** command to determine the nature of a system crash. Refer to the manpages for information about this command.

Hopefully, you will encounter crashes rarely, but if your system does crash and is unable to recover automatically and reboot itself, then follow these steps from the diagnostic station:

1. Confirm that the hardware and all peripheral devices are connected.

2. Power up the hardware, if necessary. See "Powering Up the Paragon™ Hardware" on page 2-3 for information on how to do this.

3. Monitor the hardware restart and diagnostic operations. Refer to the *Paragon™ System Diagnostic Reference Manual* for information about your hardware, and instructions for interpreting diagnostic output.

   - If the diagnostic tests are unsuccessful, indicating hardware errors, contact SSD Customer Service. Since hardware errors are a serious problem and can create further chaos, do not continue or try to bypass the defective hardware.

   - If the diagnostics test is successful, indicating that no hardware errors were encountered, the system displays a prompt (`DS#`).

4. Decide which start-up mode you want to initiate.

   - If you want the system to deny access to all users but *root*, plan on working in single-user mode. After a crash, it is wise to work in single-user mode initially. You will want to perform post-crash administrative operations before enabling system access to other users.

- If you want the system to allow access to you and to all other users with login permission, plan on working in a multi-user mode.

Note that you must boot to multi-user mode in order to perform tasks that require access to nodes other than the boot node, such as checking PFS file systems. In this case, you should follow the procedures in the section "Shutting Down the System From Multi-User Mode" on page 3-2, then boot to multi-user mode using the procedures in "Changing User Modes" on page 4-14.

5. Enter the **reset** command that corresponds to the desired start-up mode, as described in "Booting the System" on page 4-1.

Identifying the System Run Levels

A *run level* specifies the state of the system, and defines which processes are allowed to run in that state. There are four run levels:

0	Represents the halt state.
S (or s)	Represents single-user state.
2	Represents a multi-user state without network services.
3	Represents a multi-user state with network services.

The *inittab* file contains line entries that define the specific run levels and the *rc* (run control) scripts that are associated with the run level. When the **init** process starts, it reads the *inittab* file and executes the relevant run control scripts. The scripts, in turn, define which processes are to run (and which processes are to be killed if the system is changing from one level to another) at a specific run level. Refer to Chapter 9 in this guide for information about reading and modifying the *inittab* file.

Changing the System Run Level

Before changing to a new run level, you should check the *inittab* file to confirm that the run level to which you intend to change supports the processes you need. Of particular importance is the **getty** process, since it controls the terminal line access for the system console and other logins. Make sure that the **getty** entry in the *inittab* file allows system console access at all run levels. Refer to the *inittab* file manual page in the *OSF/1 System and Network Administrator's Reference* for more information about defining run levels. Refer to the **getty** command in the *OSF/1 System and Network Administrator's Reference* for more information about defining terminal lines and access.

Since a change in run level could result in the termination of the user's **getty** process (which disables their login capability) as well as the termination of other processes that they are running, you should communicate the change to each logged in user. Use the **wall** or **write** command to warn users before

you change the run level. Check the **getty** entry for user terminals to verify that the entries have the new run level specified in the entry. If not, request that users log off so that their processes will not terminate in response to a **kill** signal from **init**.

When the system initializes for the first time, it enters the default run level that is defined by the `initdefault` line entry within the *inittab* file. The system continues at that run level until **init** receives a signal to change run levels. The following sections describe these signals and provide instructions for changing run levels.

Changing Run Levels From Single-User Mode

Use the Bourne shell when working in single-user mode, and initiate a run level change by pressing `<Ctrl-d>`. The shell terminates in response to the signal, and **init** takes you to run-level 3.

Changing Run Levels From a Multi-User Mode

When the system is running at one of the two multi-user run levels, you can initiate a run level change by invoking the **init** command. To use the command, log on as *root* and enter the **init** command using this syntax:

init [**0** | **s** | **2** | **3** | **q**]

where:

0	Specifies a halt state.
s	Specifies the single-user run level.
2	Specifies the multi-user run level with local processes and daemons.
3	Specifies the multi-user run level with remote processes and daemons.
q	Specifies that **init** should reexamine the *inittab* file.

Instructing init to Change to a Different Multi-User Run Level

To instruct **init** to change from the current multi-user run level to a different multi-user run level, enter the **init** command with the multi-user run level argument that corresponds to the new run level that you want to enter. For example, to change from run level **2** to run level **3**, enter:

```
init 3
```

In response to your entry, **init** reads the *inittab* file, and follows the instructions that correspond to the change in run level.

NOTE

Once you change from run level **2** to run level **3**, you cannot return to run level **2** unless you shutdown the system gracefully and reboot the system using the instructions in "Booting a Running Paragon System With Reset" on page 4-13.

Instructing init to Change to the Single-User Run Level

To instruct **init** to change from the current multi-user run level to the single-user run level, enter the **init** command with the **s** run level argument. For example, to change from the current run level to the single-user run level, enter:

```
init s
```

Instructing init to Reexamine the *inittab* File

To instruct **init** to reexamine the *inittab* file, enter:

```
init q
```

In response, **init** reexamines the *inittab* file and starts new processes, if necessary. For example, if you have recently added new terminal lines, **init** activates the **getty** process for these terminal lines in response to the **init q** command. Refer to the **getty** command for more information about the relationship between terminal lines and the **init** command.

Setting and Resetting the System Clock

The system has an internal clock that you set when you install the system. The clock maintains the time and date whether the power is on or off. Nevertheless, there are occasions when you might need to reset the time or date. For example, with battery-powered clocks, you might need to reset the time as a result of battery failure. Or you may need to synchronize system time with standard time. To set the date and time, log in as *root* and enter the **date** command using this syntax:

date *yymmddhhmm.ss*

where:

yy	Year as a two-digit integer.
mm	Month as a two-digit integer.
dd	Day as a two-digit integer.

hh	Hour as a two-digit integer, using a 24-hour clock.
mm	Minutes as a two-digit integer.
ss	Seconds as a two-digit integer.

For example, to set the date and time to 1:02:47 PM on March 21, 1993, enter this command:

```
date 9303211302.47
```

Do not reset the clock while other users are logged in to the system. Likewise, do not reset the clock to an earlier date or time than the current (real) time. For more information about the **date** command, refer to the corresponding manual page in the *OSF/1 Command Reference*.

Setting the Network Time Daemon (NTP)

The Network Time Daemon (NTP) coordinates a consistent clock time between all the nodes in the system. You can configure the Network Time Daemon at any time by running the **postboot** installation script.

In order to configure the Network Time Daemon you must have *root* login privileges and you must know the name of the Network Time Server (the name of the computer that maintains the network-wide clock). With these two requirements satisfied, use the following procedure:

1. Log into the Paragon system as *root*.

```
login -l root <system>
Password: <root_password>
```

2. Execute the following commands to define and configure the Network Time Daemon.

```
% cd /
% postboot
```

You will receive the following display asking you to update the message of the day.The display will also query you as to which tasks you want to perform:

```
Postboot Process

This script will perform final configuration of your
Paragon system.

Updating message of the day (motd) ...

Is this correct date/time: <date_and_time> (y/n)? [y]
y

The remaining tasks this script will perform are:

  - Configure Network Time Daemon
  - Install compilers and/or documentation
  - Install Paragon tools
  - Create the terminfo databases
  - Create the spell dictionaries
  - Create the cat directories
  - Create the lint libraries
```

3. When it asks if you wish to configure the NTP, answer *y* and continue:

```
Do you wish to proceed with configuring Network Time Daemon
(y/n)? [y]

y
```

4. Provide the name of your network time server?

```
<name_of_server>

Setting up network configuration file '/etc/ntp.conf'
Starting the network time server ...
Network Time Service started
```

For more information on the **postboot** command, refer to the Manpages.

System Shutdown and Recovery 3

Shutting down the system is a routine task that you perform periodically. In most circumstances, you can shut down the system easily and with minimal disruption to other system users. This chapter shows you how to shut down and reboot the Paragon™ system.

There are several good reasons for shutting down the system:

- You need to upgrade your software, or add new hardware to your configuration.
- You suspect that your hardware problems.
- You notice that system performance is degrading rapidly or you see signs of possible file system corruption. You shut down the system in order to run the **fsck** program to fix problems or to confirm that none exist.

In each of these and similar situations, your first step should be to initiate a controlled shutdown of the system.

CAUTION

In any of the following procedures, never shut down the diagnostic station before shutting down the Paragon system. Doing so would cause a system error and incorrectly halt the system.

Shutting Down the System From Multi-User Mode

To shut down the system:

1. Log in or **rlogin** to the diagnostic station as *root*.

2. Connect to the Paragon system with the **console** command. The **console** command is a script that the **reset** script created in */usr/paragon/boot* the last time **reset** was run. **console** uses whatever the console connection was at the time and supports the **async**, **fscan**, and **scanio** console interfaces.

```
DS# cd /usr/paragon/boot
DS# ./console
          •
          •
          •
login:
```

   You may have to press a carriage return after issuing **console** to get a `login:` prompt.

3. At the `login:` prompt, log in as *root* to the Paragon system:

```
login: root
Password: (your root password)
```

4. Use the **sync** command to ensure that all disk writes are completed:

```
# sync;sync;sync
```

5. Use the command **shutdown now** to shut down all system services and terminate all users' login sessions immediately:

```
# shutdown now
Shutdown at 10:47 (in 0 minutes) [pid 198547]
#

        *** FINAL System shutdown message from root@gaia ***

System going down IMMEDIATELY


System shutdown time has arrived

INIT: SINGLE-USER MODE
```

NOTE

Remember, you cannot go back to the multi-user mode of operation unless you perform the **reset** command. Refer to "Rebooting A Shut-Down System" on page 3-4.

You can specify a reason for the shutdown if you like, and you can specify a delay time (instead of **now**) to give the users some time to finish up their work and log off; see the **shutdown** manpage in the *OSF/1 System and Network Administrator's Reference* for details.

6. Use the **umount** command with the **-A** option to unmount all the file systems:

```
# umount -A
```

7. Finally, use the **halt** command to halt the system's processors:

```
# halt
Reboot()ing.
syncing disks... 7 0 done.
System is down.  Please reset the system.
```

You will not see a prompt after the system has halted.

8. Once the **halt** command completes, return to the diagnostic station prompt by typing **~q**. (The key sequence **~.** also works, but if you are remotely logged into the diagnostic station you must enter **~~.** instead to prevent terminating the **rlogin** session.)

```
~q
DS#
```

9. The Paragon system is now down. To reboot the system from this point, see "Rebooting A Shut-Down System" on page 3-4. If you also want to shut down the diagnostic station, use the following command:

```
DS# shutdown -g0 y
```

CAUTION

Remember never to shut the diagnostic station down before shutting down the Paragon system first.

When you see the message "press any key to reboot," you can press any key to reboot the diagnostic station, or power down the system as described under "Powering Down the Paragon™ Hardware" on page 3-5.

Shutdown Summary

Here's a summary of the commands you use to shut down the Paragon system, for your reference. Commands preceded by DS# are executed as *root* on the diagnostic station; commands preceded by # are executed as *root* on the Paragon system.

```
DS# cd /usr/paragon/boot
DS# ./console
          .
          .
          .
login: root
Password: (your root password)
          .
          .
          .
# sync;sync;sync
# shutdown now
# umount -A
# halt
~q
DS#
```

To shut down the diagnostic station:

```
DS# shutdown -g0 y
```

NOTE

Remember, you cannot go back to the multi-user mode of operation unless you perform the **reset** command. Refer to "Rebooting A Shut-Down System" on page 3-4.

Rebooting A Shut-Down System

Once the system is down, you can use the **reset** command in */usr/paragon/boot* on the diagnostic station to reboot it:

```
DS# cd /usr/paragon/boot
DS# ./reset
          .
          .
          .
# <Ctrl-D>
```

When the **reset** command has completed and the Paragon system # prompt appears, enter multiuser mode by pressing **<Ctrl-D>**.

You can also tell the **reset** script to enter multiuser mode automatically, by setting the bootmagic parameter *RB_MULTIUSER* to 1 in the file */usr/paragon/boot/MAGIC.MASTER* on the diagnostic station.

Powering Down the Paragon™ Hardware

Under some circumstances, it is not only necessary to shut down the Paragon operating system, but you must also shut down the system's power. You might need to do this for the following reasons:

- The system is going to be moved.
- The air conditioning in the machine room is going to be turned off.
- The power to the system is going to be interrupted or unreliable.
- The system is going to be upgraded with new or additional hardware.

Before powering down the Paragon hardware, follow the procedure described under "Shutting Down the System From Multi-User Mode" on page 3-2.

SAFETY WARNING

To minimize the risk of electric shock and energy hazards, remove any watches, rings, bracelets, or necklaces that may be conductive before opening the rear door or any front panels. Also, to minimize accidental contact, immobilize one arm and hand by placing the hand in a pocket or behind your back before entering the service access areas behind the rear door and front panels. Assure that each movement is deliberate and do not contact any parts until a correct positive identification has been made. If there is any question in your mind about how to proceed in these locations, stop and refer to your SSD Customer Service representative before proceeding.

1. Power down the diagnostic station:

   A. The diagnostic station is located in cabinet 0, which is the rightmost cabinet when facing the cabinets from the front. Open the front door of this cabinet and swing out the diagnostic station.

   B. Turn off the power switch located on the back of the diagnostic station.

2. Open the back door of any cabinet. Notice the green power control board with red LEDs. This board has three white buttons on its upper-right corner. Press the bottom button to shut down power to the nodes.

   The system is now powered off, with the exception of its cooling fans. Under most circumstances, these fans should be allowed to remain running.

3. If *all* system power must be shut down (for example, if the system is going to be moved), locate the circuit breakers at the bottom on the back of the each cabinet and flip them down (off). Start with the highest-numbered cabinet (the rightmost cabinet when viewed from the back or the leftmost when viewed from the front) and proceed in numerical order to cabinet 0 (the cabinet containing the diagnostic station).

The system is now powered down. See "Powering Up the Paragon™ Hardware" on page 2-3 to power it back up again.

Booting the System 4

This chapter describes the booting process for the Paragon system. It first describes the hardware and software elements of the system that need booting. Then it describes the shell script **reset**, which resides on the diagnostic station and boots the Paragon system. The information is divided into the following subsections:

NOTE

The information in this chapter assumes that you are rebooting an installed operating system. For more information call the SSD Customer Service Organization (CSO).

NOTE

Refer to "Starting the System" on page 2-1 for a description of the other start up facilities. Refer also to the system Release Notes for information about installing the system software.

Understanding the Boot Process

When you boot the Paragon system, you are initiating a set of critical tasks that the system must perform in order to operate successfully. As system administrator, you may need to configure devices or software applications that require specific boot information for your system. Understanding how the system boots is information you need to know before making changes to your system configuration.

What is Needed to Boot a Paragon™ System?

Figure 4-1 shows the hardware elements that need booting. It shows several workstations with Ethernet access to a diagnostic station and a Paragon system.

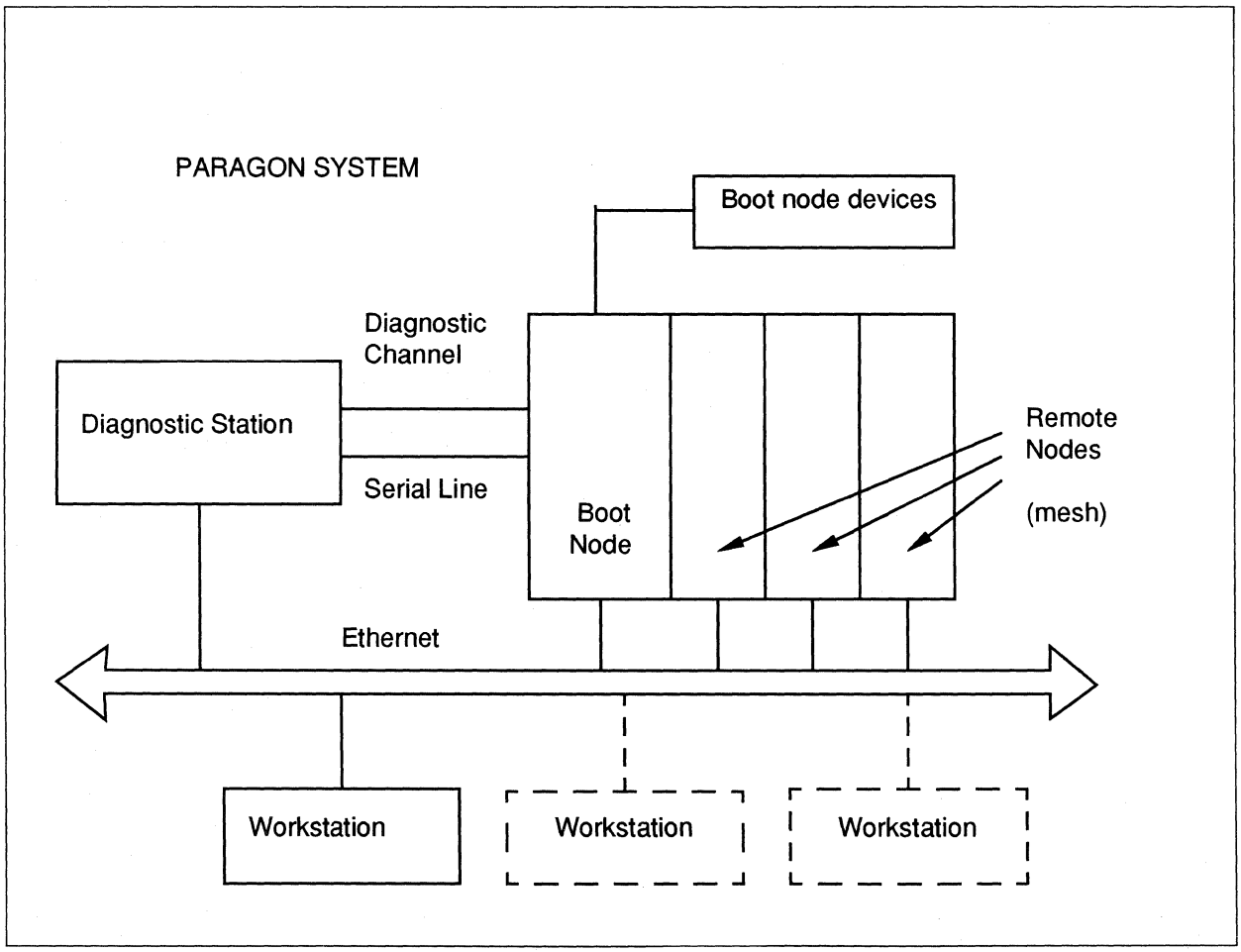


Figure 4-1. System Requirements for Booting a Paragon System

To boot the system, you must first log into a diagnostic station. All the diagnostic facilities, including the booting programs, are initiated from files on the diagnostic station. A diagnostic channel (or in some cases, a serial channel) connects the diagnostic station to the *boot node* of the Paragon system, which is the first node that boots on a operating system. The boot node is a multipurpose I/O (MIO) node located in a backplane that contains the boot node and other node boards, known as the *mesh*. The mesh includes the remaining service nodes, I/O nodes, and compute nodes. All of these nodes are booted from the boot node as part of the boot process. The boot node must have access to mass storage (a hard disk) that contains the root file system.

NOTE

Refer to "Booting a Running Paragon System With Reset" on page 4-13 for information describing the actual steps to log into the diagnostic station and boot the system.

You boot the Paragon system in the following sequence:

1. **Log in to the diagnostic station.** The diagnostic station must be up and running before you log in to the Paragon system. Powering up the hardware for both the diagnostic station and the Paragon system automatically boots the diagnostic station (see "Powering Up the Paragon™ Hardware" on page 2-3). The system administrator then logs into the diagnostic station, which provides access to the boot utilities (see "Booting a Running Paragon System With Reset" on page 4-13).

2. **Execute the reset script.** From the diagnostic station, the system administrator executes the **reset** script to download the operating system kernel the boot node. The **reset** script uses *bootmagic strings* to set various environmental descriptors in the boot node (and eventually, all other nodes). The system administrator can change the values of these bootmagic strings explicitly by changing bootmagic string values or indirectly using **reset** script options. The remaining sections in this chapter discuss the specifics of why and how to change the system environment.

3. **The boot node automatically boots the mesh nodes.** The **reset** script calls various diagnostic utilities to control mesh booting from the boot node. See "Understanding the Reset Script" on page 4-4 for a brief description of the diagnostic utilities. One of the diagnostic utilities is a boot preprocessor that uses a master copy of bootmagic strings and a hardware configuration file to generate a snapshot of the current mesh configuration and other information needed during the boot process. When the loading of the mesh completes, another diagnostic utility executes the microkernel on each mesh node and then exits. For a detailed description of the diagnostic utilities, refer to the *Paragon™ System Diagnostic Reference Manual.*

When all nodes in the mesh have successfully booted, the system is placed in the multi-user mode of operation (default) and all workstations have access to the system. For nodes that do *not* boot, the **reset** script calls on a utility that makes an entry in the *badnodes.txt* file (see "The BADNODES.TXT File" on page 4-10).

Understanding the Reset Script

The **reset** script resides on the diagnostic station in the */usr/paragon/boot* directory. The script uses a combination of command-line options and external files to set up communication with a Paragon system. The script uses external files (which we will describe later in this section) to create a *bootmagic* file. **Reset** uses the *bootmagic* file to boot the boot node on the Paragon system. The script then causes the boot node to boot the remaining nodes in the mesh, where it loads the kernel and the operating system onto each node.

How Does the Reset Script Operate?

When the system administrator enters **reset**, the script invokes several diagnostic utilities. The following list provides a high-level overview of what the **reset** script does (see Figure 4-2).

NOTE

The rectangles within Figure 4-2 show diagnostic utilities called within the **reset** script. The rounded boxes on the left show the files that **reset** uses. The rounded boxes on the right show the binary files that **reset** creates.

1. **cfgpar** combines the information from two files, *SYSCONFIG.TXT* and *DEVCONF.TXT*, to create *SYSCONFIG.BIN*, a binary which completely describes the system configuration. As system administrator, you can edit the *DEVCONF.TXT* text file if you plan to reconfigure your system. See "The DEVCONF.TXT File" on page 4-8 for detailed information.

2. **initutil** resets the Paragon system hardware, based on the contents of *SYSCONFIG.BIN*. After all nodes are reset, they each perform built-in Node Confidence Tests (NCTs), and boot a low-level system monitor stored in on-board nonvolatile flash memory. See "Using Reset Command-Line Options" on page 4-7 for the *flash* option.

3. **bootpp** combines the information from *SYSCONFIG.TXT*, *BADNODES.TXT*, and *MAGIC.MASTER* into a file called *bootmagic* that completely describes the system software configuration. See "The MAGIC.MASTER File" on page 4-10, "The BADNODES.TXT File" on page 4-10, and "The SYSCONFIG.TXT File" on page 4-9 for detailed information.

4. **scanio** opens a communication channel with the boot node. This channel becomes the system console. If your diagnostic station is communicating *asynchronously* with the Paragon system, the **reset** process uses **async** to communicate over the serial line.

5. **fscan** starts the system watchdog (if requested). The watchdog monitors node activity and optionally reboots the system whenever a node appears to be hung. See "The MAGIC.MASTER File" on page 4-10 for detailed information.

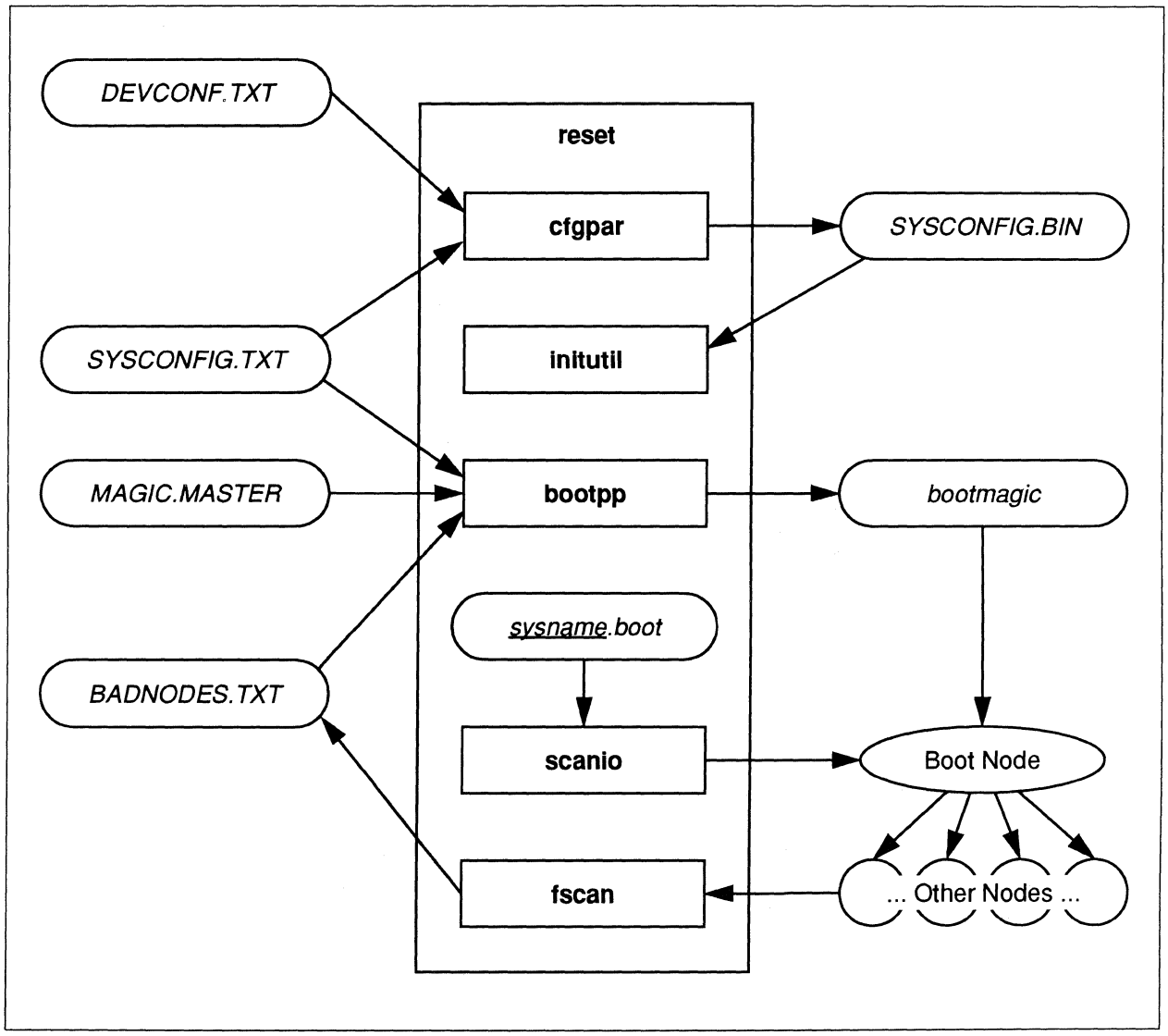


Figure 4-2. Files and Diagnostic Utilities Used By reset Script

NOTE

For detailed information about the diagnostic facilities in the above figure, refer to *Paragon™ System Diagnostic Reference Manual.*

Boot Configuration Files

The **reset** script uses the following boot configuration files in the diagnostic station in the directory */usr/paragon/boot* to boot the system. As system administrator, you can edit these files to control the way **reset** boots the system. See "Boot Configuration Files" on page 4-6 for more information

DEVCONF.TXT	List of devices attached to the system (must be present); A default file is created during system installation. You can edit this file and use it to create the *SYSCONFIG.TXT* file. See "The DEVCONF.TXT File" on page 4-8 for detailed information.
SYSCONFIG.TXT	Description of hardware configuration (must be present; can be created from *DEVCONF.TXT* file). See "The SYSCONFIG.TXT File" on page 4-9.
BADNODES.TXT	List of failed nodes (optional; can be created automatically by the system watchdog). See "The BADNODES.TXT File" on page 4-10.
MAGIC.MASTER	Contains bootmagic strings, which provide a description of the software configuration (must be present).See "The MAGIC.MASTER File" on page 4-10.

Controlling the Boot Process

You can control the boot process using one or more of the following:

- **reset** command-line options.
- Editing the system configuration files *DEVCONF.TXT*, *SYSCONFIG.TXT*, or *BADNODES.TXT*.
- Editing bootmagic strings in the *MAGIC.MASTER* file.

Using Reset Command-Line Options

Command-line options to the **reset** script (refer to Table 4-1) may appear in any order on the command line. Options are processed left to right in the order they appear in the command line (e.g., **reset autocfg**). All options consist of a single word with the exception of the **-f** option. There is minimal error checking for incompatible option combinations.

Table 4-1. Options for the Reset Script

Option	Description		
autocfg	Calls the diagnostics utilities to automatically create *SYSCONFIG.TXT* file using the *DEVCONF.TXT* file. See "The DEVCONF.TXT File" on page 4-8 for an example of this option.	**flash**	Accesses the Flash Utility, which allows you to reprogram or upgrade the Flash EPROMs in a Paragon system. Refer to the Flash Utility documentation for more information.
autoreboot	Uses the automatic reboot feature of **watchdog**. See the Manual pages.	**ignorelock**	Ignores lock on the scan device after stopping the **reset** script.
boot	Boots the system. Use this option in conjunction with the **autocfg** option.	**ramdisk**	Resets the system and boots the ramdisk facility. See the **reset** manual pages.
copy	Copies the location of a remote kernel to `BOOT_KERNEL_NAME` prior to booting the mesh.	**skip**	The system downloads the kernel and the *bootmagic* file to the boot node, but an interactive connection is not established with the console.
-f *magic_file*	Specifies a different *MAGIC.MASTER* file other than the default.	**watchdog**	Starts the **watchdog** program after the operating system resets

Editing the Configuration Files

The Paragon system requires a binary file called *SYSCONFIG.BIN*, which specifies the total hardware and software configuration needed to operate. Figure 4-3 shows the files and utilities necessary to create the *SYSCONFIG.BIN* file, representing the Paragon system configuration. When you boot the system for the first time, the **reset** script uses the system-generated *HWCFG.TXT* file and the user-generated*DEVCONF.TXT* file from the diagnostic station directory */usr/paragon/boot* to create the *SYSCONFIG.BIN* file.

The diagnostic utility **hwcfg** probes the Paragon system and determines which backplanes are accessible. It then probes each slot in each backplane and determines if the slot is empty. If the slot is not empty, it will determine the type of node boards available and how much memory is available on each. See "CBS Numbering" on page 1-10 for detailed information.Then, the diagnostic utility **mergcfg** combines the output of **hwcfg** and the file *DEVCONF.TXT* in order to create the file *SYSCONFIG.TXT*. See "The DEVCONF.TXT File" on page 4-8 for detailed information. The **reset** script then uses *SYSCONFIG.TXT* and calls **cfgpar** to create the *SYSCONFIG.BIN* file.

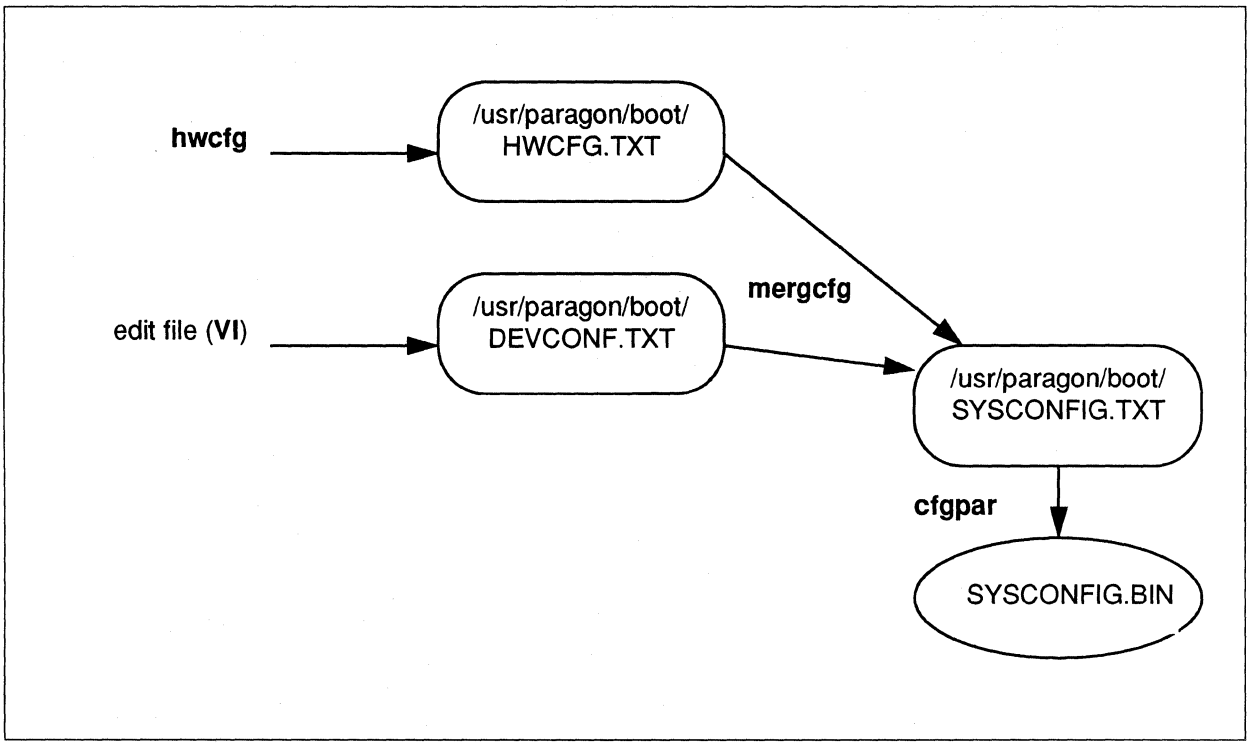


Figure 4-3. Creating a New System Configuration

If you add a device or reconfigure your system, you can edit the *DEVCONF.TXT* file and run the **reset** script with the **autocfg** option to regenerate the *SYSCONFIG.BIN* file. The next sections describe the contents of *DEVCONF.TXT* and *SYSCONFIG.TXT* files and their relationship to each other and the creation of *SYSCONFIG.BIN*

The *DEVCONF.TXT* File

During the booting process, diagnostic utilities read the *DEVCONF.TXT* file, which lists all the devices that connect to the system and to what nodes these devices connect.

The default *DEVCONF.TXT* file resides in */usr/paragon/boot* in the diagnostic station. The file is a line-oriented text file with each line representing configuration information for devices or device controllers. The first line in the file is `DEVICE`; the last line is `END_DEVICES`. Between these two configuration keywords are a series of single line entries for each device in the system.

As a system administrator, you should edit this file if you reconfigure the hardware in your system or add a device. You can then enter the following command from the */usr/paragon/boot* directory to create a new *SYSCONFIG.TXT* file and reboot the Paragon system.

```
DS # ./reset autocfg boot
```

Refer to "Booting with Reset and DEVCONF.TXT" on page 4-18 for an extended example of generating *SYSCONFIG.TXT* from *DEVCONF.TXT*.

The information in the *DEVCONF.TXT* file is necessary prior to using the **autocfg** option of the **reset** script. If this file does not exist, the system generates the following error message:

```
You need a file called DEVCONF.TXT
```

Refer to the *DEVCONF.TXT* manual pages for a detailed description of all line entries and their fields.

The *SYSCONFIG.TXT* File

This file defines how many cabinets, backplanes and nodes the system includes and what devices attach to each *node*. The boot preprocessor **bootpp** reads the file *SYSCONFIG.TXT* (as well as *MAGIC.MASTER*, and *BADNODES.TXT)* to boot the Paragon system (see Figure 4-3 on page 4-8). See "The BADNODES.TXT File" on page 4-10 and "The MAGIC.MASTER File" on page 4-10 for further information.

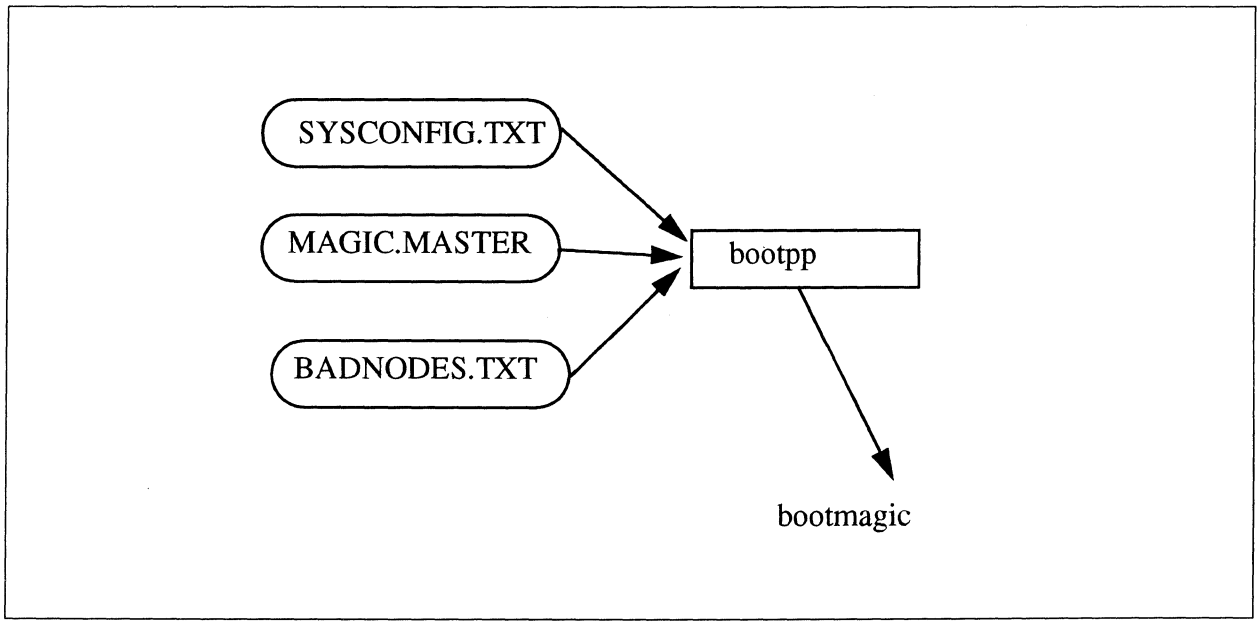


Figure 4-4. Configuration Files and Utilities

You can create the *SYSCONFIG.TXT* file automatically by first creating a file called *DEVCONF.TXT* (see page 4-8), then invoke the **reset** script with the **autocfg** option.

Each line of the file is either a device descriptor line, or a comment line. The **bootpp** diagnostic utility interprets blank lines as comment lines. Comment lines begin with a hash symbol (#). All characters to the right of the hash symbol are ignored up to the newline (CR). Device descriptor lines

begin with a keyword and are followed by one or more tokens. Tokens are single words and must not be enclosed in single or double quotes. All line types are case insensitive. For long entries, you may place a line continuation character (\) at the end of a line to continue to the next line.

There are only three device descriptor keywords: `CABINET`, `BP` and `S`. Enter the following command to see more information on the keywords and their fields.

```
% man SYSCONFIG.TXT
```

The *BADNODES.TXT* File

The *BADNODES.TXT* file contains a list of nodes which have caused three successive reboots. The default file was created during the initial boot. The system **watchdog** program automatically updates *BADNODES.TXT*, which is a line-oriented file with each line being either a comment line or a bad node line. A comment line begins with a pound sign (#). All other lines are considered bad node lines. These lines consist of the CBS, or cabinet, backplane, slot (see "CBS Numbering" on page 1-10 for more information) locations followed by a description of why the node was deemed non-operational. In the case of **watchdog** adding a node to this file, the comment will be:

```
00A02      <watchdog> Node failed 3 times
```

Any node defined in this file has the same effect as replacing GPNODE with the keyword FAILED in the *SYSCONFIG.TXT* file. It removes the node from all bootmagic strings and deactivates the processor port on the corresponding Mesh Routing Chip (MRC).

The *MAGIC.MASTER* File

This file provides *bootmagic* strings to both the Paragon operating system and the **reset** script. The filename can be changed by setting the variable *MAGIC_MASTER* or by invoking **reset** from */usr/paragon/boot* using the syntax:

```
DS# ./reset -f magic-file
```

This file is line-oriented with each line consisting of a keyword/value pair or a comment. A comment line begins with a pound sign (#). All characters to the right of the pound sign are ignored up to the newline. If a line is longer than the width of the screen it may be continued by placing a backslash(\ )character at the end of the line.

There are two kinds of *bootmagic* strings: Those intended for the Paragon system and those intended for the **reset** script. The *bootmagic* strings for the **reset** script all begin with **RST_**. The Paragon system uses all other *bootmagic* strings.

Bootmagic Strings and Reset Strings

The behavior of the **reset** script can be controlled by setting the following environment variables:

MAGIC_MASTER	Specifies the pathname of the *MAGIC.MASTER* file (default *./MAGIC.MASTER*).
SYSCONFIG	Specifies the pathname of the *SYSCONFIG.TXT* file (default *./SYSCONFIG.TXT*).
MACH_KERNEL	Specifies the pathname of the boot node kernel on the diagnostic station (*/usr/paragon/boot/mach_kernel*).

For example, to change the *MAGIC.MASTER* file from *./MAGIC.MASTER* to *./MAGIC.new*, you could set the *MAGIC_MASTER* variable as follows:

```
DS# MAGIC_MASTER=MAGIC.new
DS# export MAGIC_MASTER
```

Reset Strings

The behavior of the **reset** script can also be controlled by setting the following reset strings in the *MAGIC.MASTER* file:

RST_CONFIGURATION	Specifies the system configuration: **condo**, **multi**, or **full** (default **full**). A **condo** system has single backplane, a **multi** system has two or three backplanes, and a **full** system has four or more backplanes. (Note that the **condo** and **multi** configurations are not supported; these values are used only for internal development.)
RST_TOP_BACKPLANE	Specifies the top backplane in the system: **A**, **B**, **C**, or **D**, with **A** being closest to the floor (default **D**). This variable should be set to a value other than **D** only when *RST_CONFIGURATION* is set to **condo** or **multi**.
RST_BOOT_DIR	Specifies the directory where boot configuration files, kernels, and other files used in booting are found (default */usr/paragon/boot*).
RST_SERIAL_DEVICE	Specifies the absolute pathname of the device where the serial line from the Paragon system is attached to the diagnostic station (default */dev/tty1a*).

RST_MAGIC_MASTER	Specifies the pathname of the *MAGIC.MASTER* file (default is the value of the environment variable *$MAGIC_MASTER*, or *./MAGIC.MASTER* if the environment variable is not set). Note that this bootmagic string must be set in the default *MAGIC.MASTER* file, and does not take effect until after the default *MAGIC.MASTER* file has been parsed for *RST_* strings.
RST_SYSCONFIG_TXT	Specifies the pathname of the *SYSCONFIG.TXT* file (default is the value of the environment variable *$SYSCONFIG*, or *./SYSCONFIG.TXT* if the environment variable is not set).
RST_DEVCONF_FILE	Specifies the pathname of the *DEVCONF.TXT* file (default *./DEVCONF.TXT*).
RST_MACH_KERNEL	Specifies the pathname of the boot node kernel on the diagnostic station (default is the value of the environment variable *$MACH_KERNEL*, or */usr/paragon/boot/mach_kernel* if the environment variable is not set).

Booting Tasks

This section describes booting procedures. It presents task-oriented steps for various boot options, including the following subsections:

NOTE

Refer to the system Release Notes for information about installing the system. Refer also to the on-line Manpages for additional information about the **reset** script.

The following sections present **reset** command options that cover most booting operations. In some cases, the system administrator may need to shutdown the system, power-down, and power-up the Paragon hardware. Throughout the booting procedures, you will be sent to the following sections:

- "Shutting Down the System From Multi-User Mode" on page 3-2
- "Powering Down the Paragon™ Hardware" on page 3-5
- "Powering Up the Paragon™ Hardware" on page 2-3

Booting a Running Paragon System With Reset

In most instances, you will use the following procedure to boot (or reboot) a Paragon System that is running using **reset**:

1. Log onto the diagnostic station and enter the following commands:

```
DS# cd /usr/paragon/boot
DS# ./console
```

   The **console** command establishes communication with the Paragon system.

2. Log onto the Paragon system.

```
login: root
Password: (your root password)
```

3. Shut the Paragon system down. For a complete description of the following commands, refer to Chapter 3, "System Shutdown and Recovery."

```
# shutdown now
# halt
~q
DS#
```

CAUTION

If the system does not respond to the **shutdown** command, it may be hung or in a panic. You can reboot the system without shutting down the system. It is recommended, however, that with any of the following procedures, you should never shut down the diagnostic station before shutting down the Paragon system. Doing so would cause a system error and incorrectly halt the system.

4. Change to the boot directory on the diagnostic station:

```
DS# cd /usr/paragon/boot
```

5. Enter the **reset** command to boot the system.

```
DS# ./reset
```

This step boots the boot node and the remaining nodes on the mesh, after the *root* file system is checked and mounted. You will see the following display:

```
Paragon Reset, Version : 1.45 $
Copyright (c) 1992,1993,1994 Intel Corporation. All Rights
Reserved

Generating SYSCONFIG.BIN...
Using MAGIC.MASTER as the Magic file.
Creating bootmagic file...
Booting gaia using luna:/usr/paragon/boot/mach_kernela...
Downloading bootmagic strings...done.
.
.
.
```

Any nodes that fail to boot are reported to the console. When the **reset** command completes, you will see the Paragon system prompt.

Changing User Modes

When you boot a Paragon system for the first time, it enters the single-user mode. In this mode, only the boot node is active and only one system user (root) has access to the system. You can use the following steps to change the value of the bootmagic string `RB_MULTIUSER` to change the user mode from single-user to multi-user (or vice-verse):

1. Log into the diagnostic station and shutdown the paragon system (See step 1 through 3 of "Booting a Running Paragon System With Reset" on page 4-13).

2. Make a copy of the file */usr/paragon/boot/MAGIC.MASTER*:

```
cp MAGIC.MASTER MAGIC.MYFILE
```

3. Use your editor (such as **vi** or **ed**) to edit the *RB_MULTIUSER* entry in your file */usr/paragon/boot/MAGIC.MYFILE* to one of the following two values:

   - For single user mode, RB_MULTIUSER=0
   - For multi-user mode, RB_MULTIUSER=1 (default)

   The following list shows an example *MAGIC.MYFILE* with the *RB_MULTIUSER* entry shown in bold. Notice that the multi-user mode is selected.

```
BOOT_CONSOLE=F
BOOT_GREEN_LED=Dci
BOOT_RED_LED=Dcgl
BOOT_ARCH=paragon
RB_MULTIUSER=1
RST_CONFIGURATION=condo
RST_TOP_BACKPLANE=A
BOOT_LOAD_SYMBOLS=1
```

4. Enter the following command from the diagnostic station prompt:

```
DS# ./reset -f MAGIC.MYFILE
```

   When the **reset** command completes, the Paragon system enters the multi-user mode.

Changing Reset Environment Variables

The behavior of the **reset** script can be controlled by setting three environment variables:

MAGIC_MASTER	Specifies the pathname of the *MAGIC.MASTER* file (default *./MAGIC.MASTER*).
SYSCONFIG_TXT	Specifies the pathname of the *SYSCONFIG.TXT* file (default *./SYSCONFIG.TXT*).
MACH_KERNEL	Specifies the pathname of the boot node kernel on the diagnostic station (*/usr/paragon/boot/mach_kernela* or */usr/paragon/boot/mach_kernelb*).

For example, to change the *MAGIC.MASTER* file from *./MAGIC.MASTER* to *./MAGIC.new*, you could set the *MAGIC_MASTER* variable as follows:

```
DS# MAGIC_MASTER=MAGIC.new
DS# export MAGIC_MASTER
```

Starting the System Watchdog

The system watchdog is a utility you can invoke to monitor the status of the nodes in your system. The system watchdog can detect when a node crashes and automatically reboot your system. Use the following procedure to start the watchdog feature.

NOTE

The watchdog feature (as well as bad nodes information) is a part of the **fscan** diagnostic utility. The following procedure assumes you are using the console interface **fscan** between the diagnostic station and the Paragon system. You can also set the system watchdog using the **async** serial interface. See the **fscan** and **async** Manual pages for a complete description.

1. Log into the diagnostic station and shutdown the Paragon system (see steps 1 through 3 of "Booting a Running Paragon System With Reset" on page 4-13).

2. Change to the boot directory.

   ```
   DS# cd /usr/paragon/boot
   ```

3. Edit the *MAGIC.MASTER* file in */user/paragon/boot* directory of the diagnostic station and change the `BOOT_CONSOLE` entry with the following command:

   ```
   BOOT_CONSOLE=f
   ```

   This entry boots the system using the **fscan** interface.

4. Edit the *fscan.cfg.* file in */user/paragon/boot* directory of the diagnostic station and set the polling entry value to the following value:

   ```
   set polling on
   ```

   This allows the **fscan** interface to enable the watchdog feature.

5. Enter the following command from the diagnostic station prompt:

```
%./reset
```

When the **reset** command completes, you will see the Paragon system prompt. The system will now automatically reset when the system watchdog detects a node crash.

6. Enter the following commands to change to the */sbin* directory on the system and change ownership of the **watchdog** command. This will ensure that no one besides *root* can change the watchdog feature:

```
%cd /sbin
%chown root watchdog
```

Specifying the Debug Kernel

If the **debug** option is specified in the **reset** command line, the **reset** script boots the debug version of the *mach_kernel*:

```
DS# reset debug
```

With this command, the kernel *mach_kernel.db* located in the directory */usr/paragon/boot* of the diagnostic station will download to the boot node.

NOTE

The debug version of the *mach_kernel* displays more information than the default *mach_kernel*, especially in the display of errors. Refer to the **autoddb** manpage for information describing the output display that you receive while using the debug kernel.

Disabling the Mesh

In some cases, you may only want to boot the boot node but not the remainder of the mesh. This differs from single-user mode in that the UNIX operating system still enters the multi-user mode, even though the mesh is not booted. Use the following steps to disable the mesh:

1. Log into the diagnostic station, shutdown the Paragon system, and change to the `/usr/paragon/boot` directory on the diagnostic station.

2. Use your editor to edit the `DISABLE_BOOTMESH` entry in the file */usr/paragon/boot/MAGIC.MASTER* to the following value:

```
DISABLE_BOOTMESH=1
```

3. Enter the following command from the diagnostic station prompt:

```
DS# ./reset
```

When the **reset** command completes, you will see the Paragon system prompt.

Booting with Reset and *DEVCONF.TXT*

By editing the *DEVCONF.TXT* file and running the **reset** script, you can quickly create software and hardware configuration files after:

- Reconfiguring cabinets and backplanes
- Installing I/O interface boards
- Installing Ethernet devices
- Installing disk, tape, RAID, or paging devices

NOTE

Refer to the *Paragon™ System Hardware Installation Manual* for information about each of the above hardware procedures. These procedures may also involve setting various environment variables or *bootmagic* strings *before* running the **reset** script. Call the SSD Customer Service Organization for assistance.

The **reset** script uses the edited *DEVCONF.TXT* file to generate a new hardware and system configuration and then boots the system with the new configuration. After installing a device, you would then perform the following steps:

1. Power up the diagnostic station and the Paragon system. Refer to "Powering Up the Paragon™ Hardware" on page 2-3.
2. Log into the diagnostic station and change directories to */usr/paragon/boot*.

3. Edit the *DEVCONF.TXT* file in the */user/paragon/boot* directory. For example, the following *DEVCONF.TXT* file shows the entry lines (in bold) that you would enter to add a MAXTOR disk and its associated MIO board to the configuration. Refer to "Configuring Additional RAID Subsystems" on page 11-14 for a detailed example:

```
DEVICES
ENET 00A00
DISK 00A00 ID 0 MAXTOR
MIO 00A00 H04
ENET 00A15
DISK 00A15 ID 0 MAXTOR
MIO 00A15 H04
```

4. Enter the following command to generate the new system configuration:

```
DS# ./reset autocfg
```

The **reset** script uses the edited *DEVCONF.TXT* file to generate the following new *SYSCONFIG.TXT* file. In turn, the script then uses the *SYSCONFIG.TXT* file to provide the Paragon system with the changed hardware configuration.

```
CABINET 0
PC AP04
LED AB14

BP A AC02
S  0 GPNODE AN00 32 MRC 04 NIC B02  ENET MIO H04 CTLR0 MAXTOR
S  1 EMPTY
S  2 EMPTY
S  3 MPNODE AG00 64 MRC 04 NIC B02
S  4 MPNODE AG00 64 MRC 04 NIC B02
S  5 GPNODE AK00 16 MRC 04 NIC B02
S  6 GPNODE AK00 16 MRC 04 NIC B02
S  7 GPNODE AK00 16 MRC 04 NIC B02
S  8 GPNODE AK04 16 MRC 04 NIC B02
S  9 GPNODE AK00 16 MRC 04 NIC B02
S 10 GPNODE AK00 16 MRC 04 NIC B02
S 11 MPNODE AG00 64 MRC 04 NIC B02
S 12 MPNODE AG00 64 MRC 04 NIC B02
S 13 EMPTY
S 14 EMPTY
S 15 MPNODE AG00 64 MRC 04 NIC B02  ENET MIO H04 CTLR0 MAXTOR
```

5. Now enter the following command to boot the system with the new system configuration:

```
DS# ./reset
```

Since the **reset** script command-line processes its arguments sequentially, you could have entered the following command string in step 4 to reconfigure *and* boot:

```
DS# ./reset autocfg boot
```

Changing Default *MAGIC.MASTER* Strings

You can control the operation of the **reset** script by defining strings in the *MAGIC.MASTER* file in the */usr/paragon/boot* directory of the diagnostic station. This facility exists to minimize the editing of the **reset** script itself. The **reset** script uses the *MAGIC.MASTER* file, which can contain many boot strings.

The *MAGIC.MASTER* file must contain the following default lines, which are set during installation:

```
BOOT_FIRST_NODE=boot_node_number
BOOT_CONSOLE=cm
BOOT_GREEN_LED=Dci
BOOT_RED_LED=Dcgl
```

We recommend that you copy the *MAGIC.MASTER* into your own file (for example, *MAGIC.MYFILE*) to make changes to its values. Use the following procedure to change bootmagic values:

1. Log into the diagnostic station and shutdown the paragon system (See step 1 through 3 of "Booting a Running Paragon System With Reset" on page 4-13).

2. Make a copy of the default file */usr/paragon/boot/MAGIC.MASTER*. For example:

   ```
   cp MAGIC.MASTER MAGIC.MYFILE
   ```

3. Edit the entries in your *MAGIC.MYFILE*. You can either change existing entries or add new ones. Check the Manpage for the bootmagic command to obtain a complete listing of acceptable bootmagic strings. A few of the most often used strings are shown in Table 4-2.

   The following list shows an example *MAGIC.MYFILE* with the addition of the `BOOT_CONSOLE` entry (shown in bold) with its value set to ***F*** for **fscan** interface to the Paragon system.

   ```
   BOOT_CONSOLE=F
   BOOT_GREEN_LED=Dci
   BOOT_RED_LED=Dcgl
   BOOT_ARCH=paragon
   RB_MULTIUSER=1
   RST_CONFIGURATION=condo
   RST_TOP_BACKPLANE=A
   BOOT_LOAD_SYMBOLS=1
   ```

NOTE

Notice the prepended RST_ variable names. These values affect environment variables only in the diagnostic station.

4. Enter the following command from the diagnostic station prompt:

```
DS# ./reset -f MAGIC.MYFILE
```

The command calls on the **reset** script to use *MAGIC.MYFILE* to change values of the edited bootmagic strings. Use the **getmagic** command to display all your bootmagic strings. Refer also to the manpages for a description of all the *bootmagic* strings

Table 4-2. Common Bootmagic Strings

Bootmagic String	Description
ALLOCATOR_NODE	Specifies the node where the allocator runs.
BOOT_ALT_KERNEL_NAME	Specifies the pathname of the kernel that is broadcast to the nodes listed in the BOOT_ALT_NODE_LIST bootmagic string. The default is */mach_servers/sunmos*.
BOOT_ALT_NODE_LIST	List of nodes to run the alternate operating system on.
BOOT_COMPUTE_NODE_LIST	Specifies the list of nodes that receive the *light* server and the kernel. The list is computed during the boot.
BOOT_COMPUTE_STARTUP_NAME	Specifies the pathname of the server that is sent to the compute nodes. The default is */mach_servers/startup.compute*.
BOOT_CONSOLE	Specifies the console that all the systems nodes use. The default is **cm**.
BOOT_FIRST_NODE	Specifies the boot node, which is the first nodes that boots on the system. The default is number 0 (zero).
BOOT_GREEN_LED	Specifies that the green LED is turned off.
BOOT_KERNEL_NAME	Specifies the pathname of the kernel that is loaded on the boot node and broadcast to the nodes on the mesh. The default is */mach_servers/mach_kernela* (or *mach_kernelb* with MP nodes).
BOOT_RED_LED	Specifies that the red LED is turned off.
PAGER_NODE	Specifies which nodes use which pager. The default is that all nodes use the boot node as their pager.
PAGE_TO	Specifies which nodes use which paging device. The default is that all nodes use the device *rz0b* as the paging device.
RB_MULTIUSER	Specifies booting the system in either single-user or multi-user mode. The default is 0 (zero).
ROOT_DEVICE_NODE	Specifies the node that contains the *root* file system. The default is the boot node.
TCP_SPACE_SIZE	Specifies socket buffer space for network connections. Refer to the Manual Pages and the *Paragon™ System High Performance Parallel Interface Manual*.

Customizing the System 5

To define and customize the system environment, you identify and modify initialization files that specify processes and run levels. The standard Paragon system software installation provides the commands, directories, and files that you need to get started. In particular, you have the default files that define the available run levels and processes associated with each run level. You can change or customize the system environment easily when the need arises by using these files as templates.

This chapter describes the software and provides instructions for identifying, using, and modifying the files that initialize and control the system environment.

System Initialization Files

The following system files and directories on the Paragon system influence system startup and operation:

/etc/inittab — One of the key initialization files whose entries define run levels and associated processes. "The /etc/inittab File" on page 5-3 describes this file.

/sbin/bcheckrc — A system initialization **rc** (run command) script associated with boot time file system checking and mounting. "Specifying the Bootwait Run Levels" on page 5-5 describes this file.

/sbin/init.d — The initialization directory containing executable files associated with system startup and the available run levels. "The init.d Directory" on page 5-7 describes the directory structure and contents.

/sbin/rc#.d — A set of individual directories that correspond to the various run levels. Each directory contains linked files that the system acts on when starting or changing a particular run level. "The rc0.d Directory and rc0 Script" on page 5-8, "The rc2.d Directory and rc2 Script" on page 5-9, and "The rc3.d Directory and rc3 Script" on page 5-10 describe the *rc* directory structure and contents.

/sbin/rc# The script that corresponds to a particular run level. There are three */etc/rc#* scripts available: **/etc/rc0**, **/etc/rc2**, and **/etc/rc3**. The same sections that describe the */sbin/rc#.d* directories describe the contents and use of these scripts.

/usr/sbin/getty The executable file that sets and manages terminal lines. "Specifying the Console Run Levels" on page 5-6 and "Specifying Terminals and Terminal Run Levels" on page 5-6 describe this program. See also the **getty** command in the *OSF/1 System and Network Administrator's Reference* for additional information.

/etc/gettydefs The file used by **getty** that contains entries to identify and define terminal line attributes. See the *gettydefs* file in the *OSF/1 System and Network Administrator's Reference* for information.

*/var/spool/cron/crontabs/** The directory containing files that hold entries to identify and define the regular or periodic activation of specific processes. See the **crontab** command in the *OSF/1 Command Reference* for more information.

*/var/spool/cron/atjobs/** The directory containing files that hold entries to identify and define the once-only activation of specific processes. Refer to the **at** command in the *OSF/1 Command Reference* for information about this file.

There are other configuration files in the Paragon system. For example, the files */etc/fstab* and */etc/pfstab* determine your file system configuration. See Chapter 9 and Chapter 10 for more information on how to work with these files.

In order to understand and take full advantage of the available functionality, you should familiarize yourself with the initialization program and the specific files and commands associated with **init**. Refer to **init** in the *OSF/1 System and Network Administrator's Reference* for a description of the program and its behavior.

Modifying the System Initialization Files

Before making any changes to the system initialization files, you should examine the default setup, evaluate the needs of your system, and make a copy of the entire set of default files. Taking precautions is always wise when making changes to system files or to files that alter the working environment. If you discover that your modifications do not create the environment that you intended, you can reinstate the default files while you work out the problems.

The */etc/inittab* File

One of the first actions taken by the **init** program is to read the */etc/inittab* file. The *inittab* file supplies the **init** program with instructions for creating and running initialization processes. The **init** command reads the *inittab* file each time init is invoked. The file typically contains instructions for the default initialization, the creation and control of processes at each run level, and the **getty** process that controls the activation of terminal lines.

The *inittab* file is composed of lines, each with four fields. Each field is separated by a colon. There is no limit to the number of lines in the *inittab* file. The fields and syntax for entries are as follows:

Identifier:*Runlevel*:*Action*:*Command*

Indentifier A one to fourteen-character field that uniquely identifies an object.

Runlevel A one to twenty-character field that defines the run levels in which the object is to be processed. *Runlevel* corresponds to a configuration of processes in a system. Each process spawned by the **init** command is assigned one or more run levels in which it is allowed to exist. The run levels are represented by the characters **0**, **2**, **3**, **S**, and **s**. The *Runlevel* field can define multiple run levels for a process by selecting more than one run level in any combination of *Runlevel* characters.

Action A one to twenty-character field that tells **init** how to treat the specified process. Some of the most common actions that **init** recognizes are:

respawn If the process does not exist or dies, **init** starts it. If the process currently exists, **init** does nothing and continues scanning the *inittab* file.

wait When **init** enters a run level that matches the run level of the entry, it starts the process and waits for its termination. As long as **init** continues in this run level, it does not act on subsequent reads of the entry in the *inittab* file.

bootwait When **init** first executes and reads the *inittab* file, it processes the **bootwait** line entry. **init** starts the process, waits for the process to terminate and, when it dies, does not restart the process.

initdefault A line with this action is processed when **init** is originally invoked. **init** uses this line to determine which run level to start from. It does this by taking the highest run level specified in the *Runlevel* field and using that as its initial state. If the *Runlevel* field is empty, this is interpreted as **0s23**, so **init** will enter run level 3. If **init** does not find an **initdefault** line in the *inittab* file, it requests an initial run level from the operator.

Additional action keywords are available and recognized by **init**. Refer to the *inittab* file in the *OSF/1 System and Network Administrator's Reference* for details.

Command This is a one to 1024-character field which holds the **sh** command to be executed. The entry in the *Command* field is prefixed with **exec**. Any legal **sh** syntax can appear in the *Command* field. Comments can be inserted with the *#comment* syntax. The line continuation character (\) may be placed at the end of a line.

Default Line Entries in the */etc/inittab* File

The operating system software provides you with a basic */etc/inittab* file that contains line entries for the most common and necessary initialization processes. For example, the */etc/inittab* file available with the operating system looks something like this:

```
is:3:initdefault:
ss:Ss:wait:/sbin/rc0 shutdown < /dev/console > /dev/console 2>&1
s0:0:wait:/sbin/rc0 off < /dev/console > /dev/console 2>&1
fs:23:bootwait:/sbin/bcheckrc < /dev/console > /dev/console 2>&1
update:3:respawn:/sbin/update > /dev/console 2>&1
s2:23:wait:/sbin/rc2 < /dev/console > /dev/console 2>&1
s3:3:wait:/sbin/rc3 < /dev/console > /dev/console 2>&1
cons:1234:respawn:/usr/sbin/getty console console
```

If you intend to change or add entries, make certain that you are familiar with the function and contents of the associated files and **rc** scripts. The following sections provide information that will help you.

Specifying the Initialization Default Run Level

At boot time, **init** looks in the *inittab* file for the **initdefault** keyword for the definition of the run level to enter. If there is no entry for **initdefault**, the system prompts you for a run level. In the example shown previously, the run level for **initdefault** is set to 3, which is the multi-user with network services state.

```
is:3:initdefault:
```

Specifying the Bootwait Run Levels

The **init** program reads the *inittab* file for the **bootwait** entry. In the previous *inittab* file sample, line 4 depicts the **bootwait** entry:

```
fs:23:bootwait:/sbin/bcheckrc < /dev/console > /dev/console 2>&1
```

In this case, **init** first invokes the */sbin/bcheckrc* script for the **bootwait** entry. Processes associated with this entry execute at run levels 2 and 3. Input comes from the system console (`< /dev/console`); both system and process error messages are sent to the console (`> /dev/console 2>&1`).

The **bcheckrc** script contains procedures associated with file system checking and mounting. It checks filesystems in three stages:

1. Uses **fsck** to check the root and *usr* file systems and the boot node's vnode paging filesystem (*/etc/fstab* passes 1 and 2). Once they have been checked, mounts those filesystems.

2. Boots the mesh, making sure all I/O nodes have enough time to initialize and become ready to receive device requests before other nodes attempt to page to them.

3. Initializes paging on the boot node's vnode paging filesystem; **fsck**'s, mounts, and initiates vnode paging filesystems for the remaining I/O nodes; performs a parallel **fsck** on all file systems listed in */etc/fstab*; finally, mounts all file systems listed in */etc/fstab*.

See the */sbin/bcheckrc* script for more details.

Specifying the Console Run Levels

Before you or anyone else can log in to your system, the **getty** program must set up the process that runs the **login** and **shell** programs for each terminal. Since a large portion of your initial work is done at the system console, the */etc/inittab* file contains an entry for setting up a **getty** process for this terminal. Line 8 in the previous sample *inittab* file contains the entry for the system console:

```
cons:1234:respawn:/usr/sbin/getty console console
```

This line instructs **init** to invoke the **getty** program, which sets the terminal attributes for the system console (*/dev/console*). The run level field specifies that the **getty** process executes at run levels 1, 2, 3, and 4. The **respawn** keyword tells **init** to recreate the **getty** process if the active process terminates. If the process is active, **init** does not respawn the process; if it terminated, the process is recreated.

In general, you should not modify this line unless you intend to limit the system console's access to different run levels. By placing limitations on the range of run levels for this terminal, you run the risk of disabling the system console when the system enters a run level that prohibits execution of the console's **getty** process.

Specifying Terminals and Terminal Run Levels

To enable user login at each terminal supported by your system, you must maintain support for the terminal types available at your site, and define the run level and **getty** process for each supported terminal type.

The */usr/lib/terminfo* database (a symbolic link to */usr/share/lib/terminfo*) defines the various terminal types. The *inittab* file tells **getty** the default teminal types for each TTY device found in */dev*.

The Paragon operating system supports a wide variety of terminal types. The *terminfo* database contains entries that describe each terminal type and its capabilities. The database itself is created by the **tic** program, which compiles the source files into data files. *terminfo* source files typically consist of at least one device description; the description conforms to a particular format. Refer to **terminfo**(4) for specific details on creating source files.

The */usr/lib/terminfo* directory contains the source files, each having the form *name*.**ti**. After you compile the source files with the **tic** command, it places the output in a directory subordinate to */usr/lib/terminfo*.

Various commands and programs rely on the files in these directories. Set your *TERMINFO* variable to the */usr/lib/terminfo* directory to instruct programs that rely on the database for information to look there for relevant terminal information.

Refer to the **getty** command and the *inittab* file format in the *OSF/1 System and Network Administrator's Reference* for information about managing terminal access.

Specifying Process Run Levels

Specific entries in the *inittab* file define the **rc** scripts that are to be executed when the system enters or changes to a particular run level. For example, the following *inittab* file entries specify the action to be taken by **init** at each of the available run levels:

```
ss:Ss:wait:/sbin/rc0 shutdown < /dev/console > /dev/console 2>&1
s0:0:wait:/sbin/rc0 off < /dev/console > /dev/console 2>&1
s2:23:wait:/sbin/rc2 < /dev/console > /dev/console 2>&1
s3:3:wait:/sbin/rc3 < /dev/console > /dev/console 2>&1
```

These entries are associated with the *rc* directory structure and are discussed in detail in the following sections.

The init and rc Directory Structure

The operating system provides you with an initialization and **rc** directory structure. The structure has four main components: **init.d**, **rc0.d**, **rc2.d**, and **rc3.d**. In addition, each of the *rc#.d* directories has a corresponding **rc** script.

The init.d Directory

The *init.d* directory contains the executable files associated with system initialization. Following is a sample listing:

```
allocator   inetd       lpd         nqs         rmtmpfiles  snmpd
amd         initpart    motd        ntpd        savecore    syslog
cron        kloadsrv    named       paging      sendmail    uucp
enlogin     loader      nfs         preserve    settime     xdm
inet        loadlevel   nfsmount    recpasswd   smd
```

The rc0.d Directory and rc0 Script

The **rc0** script contains commands that enable a smooth shutdown and bring the system to either a halt or a single-user state. As described previously, the *inittab* file contains entries that **init** reads and acts on when the system is shutting down to single user (level **s**) or halting (level **0**); for example:

```
ss:Ss:wait:/sbin/rc0 shutdown < /dev/console > /dev/console 2>&1
s0:0:wait:/sbin/rc0 off < /dev/console > /dev/console 2>&1
```

Notice that in both cases, the **rc0** script is the specified command. In addition to commands listed within the script itself, **rc0** contains instructions to commands found in the */sbin/rc0.d* directory. These commands are normally linked to files in the *init.d* directory. The script defines the conditions under which the commands execute; some commands run if the system is being halted while others run if the system is being shut down to single user.

By convention, files in the */sbin/rc0.d* directory begin with either the letter "K" or the letter "S" and are followed by a two-digit number and a filename. For example, output from an **ls -l** of the *rc0.d* directory would look similar to the following:

```
lrwxr-xr-x  1 root  system  17 Oct 1 05:35 K00enlogin ->../init.d/enlogin
lrwxr-xr-x  1 root  system  13 Oct 1 05:35 K05lpd ->../init.d/lpd
lrwxr-xr-x  1 root  system  15 Oct 1 05:35 K10inetd ->../init.d/inetd
lrwxr-xr-x  1 root  system  15 Oct 1 05:35 K15snmpd ->../init.d/snmp
lrwxr-xr-x  1 root  system  14 Oct 1 05:35 K20ntpd ->../init.d/ntpd
lrwxr-xr-x  1 root  system  18 Oct 1 05:35 K25sendmail ->../init.d/sendmail
lrwxr-xr-x  1 root  system  19 Oct 1 05:35 K30nfs ->../init.d/nfs
lrwxr-xr-x  1 root  system  18 Oct 1 05:35 K35nfsmount ->../init.d/nfsmount
lrwxr-xr-x  1 root  system  15 Oct 1 05:35 K40named ->../init.d/named
lrwxr-xr-x  1 root  system  14 Oct 1 05:35 K45syslog ->../init.d/syslog
lrwxr-xr-x  1 root  system  14 Oct 1 05:35 K50inet ->../init.d/inet
lrwxr-xr-x  1 root  system  14 Oct 1 05:35 K60cron ->../init.d/cron
```

During the boot process the system executes commands that begin with the letter "S." During a halt the system executes commands that begin with the letter "K." The numbering of commands in the */sbin/rc0.d* directory is important since the numbers are sorted and the commands are run in ascending order.

See the **rc0** command in the *OSF/1 System and Network Administrator's Reference* for additional information.

The rc2.d Directory and rc2 Script

The **rc2** script contains commands that enable initialization of the system to a non-networked multi-user state—run level 2. (The **rc2** script is run for both levels 2 and 3.) As described previously, the *inittab* file contains entries that **init** reads and acts upon when the system is booting or changing its state to run level 2. For example:

```
s2:23:wait:/sbin/rc2 < /dev/console > /dev/console 2>&1
```

Notice that here the **rc2** script is the specified command. In addition to commands listed within the script itself, **rc2** contains instructions to commands found in the */sbin/rc2.d* directory. These commands are normally linked to files in the *init.d* directory. The script defines the conditions under which the commands execute; some commands run if the system is booting, other commands run if the system is changing run levels.

By convention, files in the */sbin/rc2.d* directory begin with either the letter "K" or the letter "S" and are followed by a two-digit number and a filename. For example, output from a request for a long listing of the *rc2.d* directory would look similar to the following:

```
lrwxr-xr-x  1 root  system  16 Oct 1 05:35 S10recpasswd ->../init.d/recpasswd
lrwxr-xr-x  1 root  system  13 Oct 1 05:35 K00lpd ->../init.d/lpd
lrwxr-xr-x  1 root  system  15 Oct 1 05:35 K05inetd ->../init.d/inetd
lrwxr-xr-x  1 root  system  15 Oct 1 05:35 K10snmpd ->../init.d/snmpd
lrwxr-xr-x  1 root  system  14 Oct 1 05:35 K15ntpd ->../init.d/ntpd
lrwxr-xr-x  1 root  system  14 Oct 1 05:35 K20cron ->../init.d/cron
lrwxr-xr-x  1 root  system  18 Oct 1 05:35 K30sendmail ->../init.d/sendmail
lrwxr-xr-x  1 root  system  13 Oct 1 05:35 K35nfs ->../init.d/nfs
lrwxr-xr-x  1 root  system  18 Oct 1 05:35 K40nfsmount ->../init.d/nfsmount
lrwxr-xr-x  1 root  system  15 Oct 1 05:35 K45named ->../init.d/named
lrwxr-xr-x  1 root  system  16 Oct 1 05:35 K50syslog ->../init.d/syslog
lrwxr-xr-x  1 root  system  14 Oct 1 05:35 K55inet ->../init.d/inet
lrwxr-xr-x  1 root  system  18 Oct 1 05:35 S00savecore ->../init.d/savecore
lrwxr-xr-x  1 root  system  16 Oct 1 05:35 S05paging ->../init.d/paging
lrwxr-xr-x  1 root  system  14 Oct 1 05:35 S15uucp ->../init.d/uucp
lrwxr-xr-x  1 root  system  17 Oct 1 05:35 S25enlogin ->../init.d/enlogin
```

During the boot process the system executes commands that begin with the letter "S." During a halt the system executes commands that begin with the letter "K." Commands that begin with the letter "K" run only when the system is changing run levels from a higher to a lower level. Commands that begin with the letter "S" run in all cases. The numbering of commands in the */sbin/rc2.d* directory is important since the numbers are sorted and the commands are run in ascending order.

See the **rc2** command in the *OSF/1 System and Network Administrator's Reference* for more information.

The rc3.d Directory and rc3 Script

The **rc3** script contains commands that enable initialization of the system to a networked multi-user state—run level 3. As described previously, the *inittab* file contains entries that **init** reads and acts on when the system is booting or changing its state to run level 3. For example:

```
s3:3:wait:/sbin/rc3 < /dev/console > /dev/console 2>&1
```

Notice that here the **rc3** script is the specified command. In addition to commands listed within the script itself, **rc3** contains instructions to commands found in the */sbin/rc3.d* directory. These commands are normally linked to files in the *init.d* directory. The script defines the conditions under which the commands execute; some commands run if the system is booting, other commands run if the system is changing run levels.

By convention, files in the */sbin/rc3*.d directory begin with either the letter "K" or the letter "S" and are followed by a two-digit number and a filename. For example, output from a request for a long listing of the *rc3.d* directory would look similar to the following:

```
lrwxr-xr-x  1 root  system  14 Oct 1 05:35 S00inet ->../init.d/inet
lrwxr-xr-x  1 root  system  17 Oct 1 05:35 S05settime -> ./init.d/settime
lrwxr-xr-x  1 root  system  16 Oct 1 05:35 S10syslog ->../init.d/syslog
lrwxr-xr-x  1 root  system  15 Oct 1 05:35 S15named ->../init.d/named
lrwxr-xr-x  1 root  system  18 Oct 1 05:35 S20nfsmount ->../init.d/nfsmount
lrwxr-xr-x  1 root  system  18 Oct 1 05:35 S25preserve ->../init.d/preserve
lrwxr-xr-x  1 root  system  16 Oct 1 05:35 S30rmtmpfiles ->../init.d/rmtmpfiles
lrwxr-xr-x  1 root  system  13 Oct 1 05:35 S35nfs ->../init.d/nfs
lrwxr-xr-x  1 root  system  18 Oct 1 05:35 S40sendmail ->../init.d/sendmail
lrwxr-xr-x  1 root  system  14 Oct 1 05:35 S45ntpd ->../init.d/ntpd
lrwxr-xr-x  1 root  system  15 Oct 1 05:35 S50snmpd ->../init.d/snmpd
lrwxr-xr-x  1 root  system  15 Oct 1 05:35 S55inetd ->../init.d/inetd
lrwxr-xr-x  1 root  system  14 Oct 1 05:35 S57cron ->../init.d/cron
lrwxr-xr-x  1 root  system  14 Oct 1 05:35 S60motd ->../init.d/motd
lrwxr-xr-x  1 root  system  13 Oct 1 05:35 S65lpd ->../init.d/lpd
lrwxr-xr-x  1 root  system  15 Oct 1 05:02 S70mount ->../init.d/mount
```

During the boot process the system executes commands that begin with the letter "S." During a halt the system executes commands that begin with the letter "K." Commands that begin with the letter "K" run only when the system is changing run levels from a higher to a lower level. Commands that begin with the letter "S" run in all cases. The numbering of commands in the */sbin/rc3.d* directory is important since the numbers are sorted and the commands are run in ascending order.

See the **rc3** command in the *OSF/1 System and Network Administrator's Reference* for more information.

Controlling Process Fault Activity

To a certain extent, the system administrator can monitor and control process faults that may occur with any of the initialization processes (discussed in this chapter) and other ongoing system processes. By changing the value of the bootmagic string PARACORE_MAX_PROCESSES, changing core action environment variables, and rebooting you system, you can define and set the limit of processes that dump core. For instructions on how to change the values for this string, refer to "Changing Default MAGIC.MASTER Strings" on page 4-20.

The PARACORE_MAX_PROCESSES bootmagic string specifies a limit the system uses to determine how much core dump activity occurs in a given application. The format and values are:

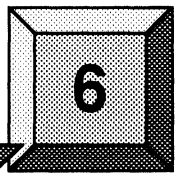
```
PARACORE_MAX PROCESSES=number_of_processes
```

where *number_of_processes* is:

1	(default) This setting limits an application to dumping the core file for the first faulting process regardless of how many processes are executing.
-1	(unlimited) This setting causes the system to depend entirely on the core action environment variables to determine how to handle faulting processes. The variables are CORE_ACTION_FIRST, CORE_ACTiON_FAULT, and CORE_ACTION_OTHER. The arguments for each are FULL, KILL, TRACE, and CONT.
> 1	This setting ignores the core action environment variables. If the total number of processes exceeds the limit, only the first faulting process dumps core. The remaining processes (faulting or not) are killed.

Controlling the amount of core dumping is especially useful in gang-scheduled environments. Refer to the bootmagic Manual pages for details of the PARACORE_MAX_PROCESSES bootmagic string.

Adding and Removing Users and Groups



Adding and removing individual users and groups is a routine but critical activity that you will be required to perform frequently. Before new users can log in successfully, they must be made known to the system. Likewise, when users or groups no longer have privileges on the system, you must remove their identity from the system.

This chapter describes the procedures, commands, and files that you use to identify and remove users and groups on your system.

Preparing to Add a New User Account

Before adding a new user account, perform the following tasks:

- Verify the existence of the file system where the user's login directory will reside. If the file system does not exist, create it now.
- Verify the existence of the group which the new user will join. If the group does not exist, create it now by following the instructions in "Adding a New Group to the /etc/group File" on page 6-11.

Once the file system and user's group exist, you can add a new user to your system. When adding a new user, you can perform the required tasks interactively or manually. The following sections describe both interactive and manual procedures.

NOTE

After adding a new user to the system, you must stop and restart the Network Queueing System (NQS) as described in the *Paragon™ System Network Queueing System Manual.* If you do not, NQS will not schedule the user's jobs.

Adding a New User Interactively

To add a new user interactively, use the **adduser** program which automates many of the tasks involved in adding a new user to your system. For example, you can add a new user account to the system password file, set up a home directory with the *.cshrc*, *.login*, and *.profile* files created for the new user, and add the user to a specific group.

To access and use the **adduser** program, follow these steps:

1. As **root**, enter the **adduser** command:

```
adduser
```

2. Respond to the prompts that the program displays. The program is simple to use, and the prompts are self-explanatory. For reference information, see **adduser** in the *OSF/1 System and Network Administrator's Reference*.

Adding a New User Manually

To add a new user manually, you perform the following tasks:

1. Add an entry in the */etc/passwd* file for the new user.
2. Modify entries in the */etc/group* file or add a new entry for the new user.
3. Create the user's login directory, supply the default shell scripts for the user's working environment and set the user's disk quotas.
4. Create the user's mail file.
5. Protect the user account by assigning a password.

The following sections describe these tasks and provide instructions for editing the files manually.

Adding a User Account to the /etc/passwd File

For every new user, you must add a line to the */etc/passwd* file. This file is a very important component of your system; it identifies each user (including root). If */etc/passwd* is inaccessible or if it gets corrupted, you risk disabling **root** and other users from logging on.

Use the **vipw** command to modify the */etc/passwd* file. The **vipw** command ensures that no other user or process can access the */etc/passwd* file while you are editing it. Before writing your changes back to the disk, **vipw** performs several consistency checks. By default, **vipw** invokes the **vi** editor.

If you prefer to use another editor, assign the name of that editor to the environment variable, EDITOR, in your *.login* (or *.profile*) file. For additional information on the **vipw** command, refer to the *OSF/1 System and Network Administrator's Reference*.

/etc/passwd File Entries

Each entry in the */etc/passwd* file is a single line that contains seven fields per line. The fields are separated by colons and the last field ends with a newline character. The following text shows the format of each entry and describes the meaning of each field:

```
username:password:UID:GID:user_info:login_directory:login_shell
```

username — The name for the new user account. The *username* must be unique and consist of from one to eight characters (bytes). Digits, and letters of your alphabet are allowed.

password — You cannot enter a password directly. Leave the password field empty or enter an asterisk (*). If the *password* field contains an asterisk (*), a login to that account is disabled. An empty password field allows anyone who knows the login name to log in to your system as that user. See "Changing a User's Password" on page 6-12 for instructions on assigning a user password with the **passwd** command. The **passwd** command encrypts the specified password and inserts it in the user's password field. Never try to edit in a password.

UID — The user ID for this account. This is an integer between 0 and 32,767 and must be unique for your system. The user ID 0 is reserved for **root**. We recommend that you assign user IDs in ascending order beginning with 100. Lower numbers are used for pseudo-users like **bin** or **daemon**.

GID — The group ID for this account. This is also an integer between 0 and 32,767. The group ID 0 is reserved for the group **root**. We recommend that you assign group IDs in ascending order beginning with 100.

user_info — This field contains additional user information such as the full username, office address, telephone extension, and private phone number. The **finger** command inserts this information in the *user_info* field. For additional information about the **finger** command, refer to the *OSF/1 Command Reference*. Users can change the contents of their *user_info* field with the **chfn** command. For additional information about the **chfn** command, refer to "Changing a User's Finger Entry" on page 6-13 and the *OSF/1 Command Reference*.

login_directory The absolute pathname of the directory where the user is located immediately after log in. The **login** command assigns this pathname to the shell variable, $HOME; users, however, can change the value of $HOME. If a user changes the value, then the home directory and the login directory are two different directories.

You create the login directory after adding a new user account to the */etc/passwd* file. Typically the username is used as the name of the login directory. For additional information on creating a login directory, refer to the **chown** command in the *OSF/1 System and Network Administrator's Reference* and to the **mkdir**, **chmod**, and the **chgrp** commands in the *OSF/1 Command Reference*.

login_shell The absolute pathname of the program that gets started after the user has logged in. Normally, a shell is started. If you leave this field empty, the Bourne shell */bin/sh* is started. For information on the Bourne shell, refer to **sh** in the *OSF/1 Command Reference*. Users can change their login shell with the **chsh** command. For additional information about the chsh command, refer to "Changing a User's Login Shell" on page 6-13 and the *OSF/1 Command Reference*.

A Sample Entry in the /etc/passwd File

```
jerry:*:201:20:Jerry Swanson,dev,x1234:/users/jerry:/bin/sh
```

The user account *jerry* has user ID 201 and group ID 20. The login directory is */users/jerry* and the Bourne shell (*/bin/sh*) is defined as the command interpreter. Since the password field contains an asterisk (*), the user *jerry* cannot log in to the system. "Changing a User's Password" on page 6-12 describes how to add a usable password to the */etc/passwd* file.

Adding a User Account to the /etc/group File

The */etc/group* file serves two purposes:

1. It assigns a name to a group ID defined in the */etc/passwd* file.
2. It allows users to be members of more than one group by simply adding the user names to the corresponding group entries.

Before adding a user account to the */etc/group* file, examine the file to verify that the group to which you intend to add the new account exists.

- If the group already exists (there is a line entry in the file for that group), then simply add the new user's name to the user field within the group's line entry.
- If the group does not exist (there is no line entry in the file for that group), then create a new line entry for the group and include the new user's name within that entry in the */etc/group* file.

To add or edit an */etc/group* file entry, open and edit the file manually.

/etc/group File Entries

Each entry in the */etc/group* file is a single line that contains four fields. The fields are separated by colons, and the last field ends with a newline character. The following text shows the format of each entry and describes the meaning of each field:

groupname:*password*:*GID*:*user1*[,*user2*,...,*userN*]

groupname — The name of the group defined by this entry. The *groupname* consists of from one to eight characters (bytes). Digits and the letters of your alphabet are allowed.

password — Leave the *password* field empty. Entries in this field are ignored.

GID — The group ID for this group. This is an integer between 0 and 32,767. The group ID 0 is reserved for **root**. The group ID must be unique.

user — The user account belonging to this group as defined in the */etc/passwd* file. If more than one user belongs to the group, the user accounts are separated by commas. The last user account ends with a newline character. The user list is often so long that it extends over several screen lines.

A user can be a member of more than one group.

Never modify the original */etc/group* file. Copy the original file and modify only the copy. When you have finished making changes, overwrite the original file with the modified file. This strategy ensures that you always have a consistent copy.

Sample Entries in the */etc/group* File

If you add a user account to an existing group, specify the username in the user field of that group's line entry. The following two line entries in the */etc/group* file specify that user *jerry* is a member of two groups: *tools* and *dep11*:

```
tools::20:rosy,peter,harold,maude,narble
depll::21:bill,rashad,susann,peter,shala,narble
```

If the group does not already exist, add a new line entry for the group in the */etc/group* file. For example, to create a new line entry for a group called software with the user *narble* as a member, you would add the following line to the */etc/group* file:

```
software::22:narble
```

Creating the Login ($HOME) Directory

Each user on your system needs a login ($HOME) directory. Use the following steps to create this directory manually:

1. Verify that the file system intended for user directories already exists before creating any login directories.
2. Change your working directory to the target location in the file system. For example, enter:

   cd /users

3. Make a directory for the user. For example, to make a directory for jerry enter:

   mkdir jerry

4. Change ownership of the directory to the user. For example, enter:

   chown jerry jerry

5. Change membership of the user to the desired group. For example, enter:

   chgrp tools jerry

6. Request a listing of the directory attributes. For example, enter:

   ls -lgd jerry

7. Read the listing and confirm that the attributes correspond to the user's needs. For example, here is output from the previous command:

   ```
   drwxr-xr-x 2 jerry tools 24 Jan 9 10:48 jerry
   ```

Providing the Default Shell Scripts

Users can customize their working environment by modifying their startup files. When a user logs in to the system, the invoked login shell looks for startup files in the login directory. If the shell finds a startup file, it reads the file and executes the commands.

With the exception of the */etc/profile* file, each startup file begins with a dot (.). Table 6-1 displays each shell, the corresponding startup file, and command control.

Table 6-1. Shells and Their Startup Files

Shell	Startup File	Command Control
/bin/csh	*.login* *.cshrc*	login shell login shell and subshells
/bin/ksh	*.profile*	login shell
/bin/sh	*/etc/profile* *.profile*	login shell login shell

The system uses these startup files to initialize local and global environment variables, shell variables, and the terminal type. The distributed software sometimes provides a set of default startup files in the */usr/local/skel* source directory. If your distribution software did not contain these files, you can create them yourself and place them in a source directory of your choice. Once these files are available, you need only to copy them to the login directory of each new user account.

To copy the startup files for a new user to the user's login directory, follow these steps:

1. Copy the startup files for each shell to the new user's login directory by entering the **cp** command. For example, to copy the startup files associated with the Bourne shell to user *smith* directory, enter:

```
cp /usr/local/skel/.profile /users/smith
```

2. Change directory to the new user's login directory and change file ownership and access permissions from **root** to the new user. For example, to make these changes to the *.login* file for user *smith*, enter this sequence of commands:

```
cd /users/smith
chmod 644 .login
chown pat .login
```

3. To confirm that the changes were made, get a long listing of *smith* files. For example, enter:

```
ls -al /users/smith
```

Creating a Mail File

The mail file must be created in the */usr/spool/mail* directory. The user name must be used as the filename for the mail file. The **mail** command writes all mail arriving for the specified username in the corresponding mail file. When a user wants to read mail, the **mail** command opens and reads from that user's mail file.

The following example illustrates the sequence of commands and output for creating a mail file for user *narble*:

```
cd /usr/spool/mail
touch narble
chown narble narble
chgrp tools narble
chmod 600 narble
ls -lg narble
-rw------- 1 narble tools 0 Jan 11 17:54 narble
```

The last line in the previous example specifies that user *narble* owns the mail file, he has read/write permission for it, he belongs to the tools group, and the file was created on 11 January. Once the file exists, narble can read incoming mail messages and delete the ones that he does not want to keep. With the exception of *root*, only *narble* has access to this file.

Assigning an Initial Password

Use the **passwd** command to assign an initial password for a new user account. When you enter the command, the program prompts for the password. Each password must have at least six characters (bytes) but not more than eight, and can include digits, symbols, and the letters of your alphabet. After you enter the password, the program prompts you to retype it. The second entry serves as verification.

To assign an initial password, follow these steps:

1. Enter the **passwd** command using this syntax:

   ```
   passwd username
   ```

2. In response to the program's prompt, enter the new password for the user. For example, the program displays these prompts:

   ```
   New password:
   Retype new password:
   ```

   The echo is disabled while you enter the password, thus ensuring password confidentiality. Be sure to tell the user what the password is.

See the *OSF/1 Command Reference* for a description of the **passwd** command.

Removing a User

There are several tasks that you perform and several files that you edit when you remove a user from your system. You must:

- Remove the user's files and directories.
- Remove the user's entry from the */etc/group* file.
- Remove the user's entry from the */etc/passwd* file.

The following sections describe each task and provide instructions for editing the files manually.

Removing the User's Files and Directories

Before removing anything that belongs to the user, follow these steps:

1. Make sure that the associated files and directories are no longer being used by other users on your system.
2. Back up the user's login directory to diskette or tape.

To remove the user's login directory with all of its files and subdirectories, use the **rm -r** *login_dir* command. For example, to remove the login directory and its entire tree substructure for user *mary*, enter:

```
rm -r /users/mary
```

To remove the user's mail file, use the **rm** *mail_dir* command. For example, to remove user *mary*'s mail file, enter:

```
rm /usr/spool/mail/mary
```

Make sure that there are no files left that were owned by the user. To check this, use the **find** command. The **find** command locates user files that are either links (identified by a notation of 1), user files within directories (identified by a notation of 1), or user directories (identified by a notation of 2).

If your search locates any user files or directories, use the **chown** command to change the file or directory ownership to a different user (one who still needs to access the file). If you have no reason to save or maintain these files, then remove them.

Removing the User's Account from the */etc/group* File

Since users can be members of more than one group, you must modify all line entries in the */etc/group* file that contains the username within the *user* field. However, you should always create a copy of the */etc/group* file before you modify it.

Removing the User's Account from the /etc/passwd File

After you perform this task, the user account vanishes and the system no longer has a means of identifying the user.

To remove the user's account, simply delete the line entry in the */etc/passwd* file that identifies the user. Use the **vipw** command to edit the */etc/passwd* file.

If you maintain accounting on a monthly basis, do not remove the user's line entry from the */etc/passwd* file until the monthly accounting has been done. Since the accounting commands access the */etc/passwd* file, removing the user entry would create inaccuracies in your accounting.

However, since your primary goal is to restrict the user from gaining access to the system, you can immediately suspend the user from logging in. To do this, edit the */etc/passwd* file and substitute an asterisk (*) for the encrypted user password.

Adding and Removing Groups

Whenever you add or remove a group, you must modify the */etc/group* file. There are two primary reasons for grouping user accounts:

1. Several users work together on the same files and directories; grouping these users together simplifies file and directory access.

2. Only certain users are allowed access to system files or directories; grouping these users together simplifies the identification of those privileged users.

The following sections tell you how to add and remove groups and which commands to use.

Adding a New Group to the */etc/group* File

When you want to add a new group, you must add a new line entry within the */etc/group* file. Before adding a new group manually, you need to make some decisions. For example, you must have answers to the following questions:

- What will you name the group? The group name must be unique.
- What number will you assign as the group ID (GID)? The number must be unique (within the */etc/group* file).
- When can you include this information within the */etc/passwd* file?

When you have answers to these questions, you can proceed with the actual task. To add a new group to the */etc/group* file manually, follow these steps:

1. Change directory to the */etc* directory.
2. As **root**, copy the */etc/group* file with the **cp** command. For example, enter:

   ```
   cp /etc/group /etc/group.new
   ```

3. Open the new file and add the required line entry. See "/etc/group File Entries" on page 6-5 for a listing of required fields within each line entry in the */etc/group* file.
4. Close the new file and copy it by overwriting the original */etc/group* file. For example, enter:

   ```
   cp /etc/group.new /etc/group
   ```

5. Edit the */etc/passwd* file to include the new group identification number within the *GID* field of each user who is a member of the group. Refer to "/etc/passwd File Entries" on page 6-3 for a description of the */etc/passwd* fields.

For example, to add a new group called editors to your system, add the following line to the */etc/group* file:

```
editors::50:
```

This entry is valid if the group name *editors* does not already exist (and is therefore unique within the file), and if the group ID (50) is unique and is the next ascending number available for an entry in the */etc/group* file.

Removing a Group

To remove a group that no longer has any members, delete the corresponding line from the */etc/group* file.

To remove a group that still has members, follow these steps:

1. Edit the */etc/passwd* file line entry for each member of the group. You can either assign a new group number or delete the current group number. If you assign a new group number, make sure that it corresponds to a current (or new) group entry in the */etc/group* file.

2. Remove the original group line entry from the */etc/group* file.

Changing Entries in the */etc/passwd* File

There are specific commands available to help you change the entries specified in the */etc/passwd* file. The following sections describe the commands, and explain when and how to use them.

Changing a User's Password

Sometimes a user changes but then forgets the new password. If this happens, you can change the password for the user by becoming **root** and following these steps:

1. Enter the **passwd** command using this syntax:

```
passwd username
```

2. In response to the program's prompt, enter the new password for the user. For example, the program displays the following prompts:

```
New password:
Retype new password:
```

The echo is disabled while you enter the new password, thus ensuring password confidentiality. Be sure to tell the user that you have changed the password and let him know what the new password is.

See the *OSF/1 Command Reference* for a description of the **passwd** command.

Changing the root Password

You can change the **root** password by following the same steps as described in "Changing a User's Password" on page 6-12. It is a good practice to change the **root** password periodically to protect the system from access by system users who should not have **root** access, as well as from external intruders.

Changing a User's Finger Entry

To change a user's **finger** entry, use the **chfn** command with the following syntax:

```
chfn username
```

To change your own **finger** entry, omit the *username* argument:

```
chfn
```

In response to the command, the **chfn** program displays general information and specific prompts. Simply respond to each prompt with the new information or accept what is displayed by pressing **<Return>**. The following example illustrates an interactive session.

```
chfn jerry
Default values are printed inside of '[]'.
To accept the default, type <Return>.
To have a blank entry, type the word 'none'.

Name [Jerry Swanson]:  <Return>
Office or Mail Stop (no colons or commas) []:  <Return>
Office Phone (Ex: 1234 or 123-4567) [1313]: 3311
Home Phone (Ex: 123-4567 or 123-456-7890) []: <Return>
```

Changing a User's Login Shell

To change a user's login shell, use the **chsh** command with the following syntax:

```
chsh username [ shell ]
```

For example, to change user *jerry*'s login shell from the Bourne to the C shell, enter:

```
chsh jerry /bin/csh
```

Backing Up and Restoring Files 7

No matter how careful users are they sometimes lose files. Files may be lost or corrupted by accidental user actions, program errors, and system failures. One of the more common tasks a system administrator has is helping users recover lost or corrupted files. To perform that task effectively, you must set up good procedures for backing up files at frequent and regular intervals. This chapter describes a method and the tasks that you need to perform to back up and restore files on your system.

- Choosing which file systems to back up.
- Choosing a backup schedule.
- Backing up a file system.
- Restoring a file system.
- Restoring a file.
- Creating a backup script.
- Cleaning the 4mm DAT tape drive.

Choosing Which File Systems to Back Up

It is important that all the files on your system be protected from loss: the data files as well as the system files required to run your system. Because of this you should back up your entire system including the system software at least once. Most of those system files are static. That is, once they are installed they do not often change, therefore they do not need to be backed up as frequently as data files, which change constantly. Incremental backups are also possible.

Although several file systems may be backed up simultaneously (assuming, of course, that your system has more than one backup device), each file system backup is done as a single process. To make the backup process easier, you should organize your file systems so that changing data files are on file systems that are regularly backed up, and that static files are on file systems that are backed up only occasionally. You can simplify this division by copying isolated changing files from file systems that are not periodically backed up to periodically backed up file systems just prior to performing a backup. This allows the dynamic files to be backed up without requiring an entire file system backup. You could write a shell script to do this.

Other backup strategies are possible. For example, you could use a **find** command to produce a list of files the must be backed up and then pipe this list to a backup program such as **tar** or **cpio**. This chapter describes a backup strategy using the **dump** and **restore** group of backup commands.

Pick the files systems that are regularly backed up by evaluating the likelihood that their files will change in the backup period. File systems that are constantly changing, such as the user file system, should obviously be backed up often.

Choosing a Backup Schedule

Deciding how often to back up each file system is a difficult task. You need to balance the potential loss of user time and data against the time it takes you to perform backups. You should ask the question, "How much information can I afford to lose?" The answer tells you what your minimum backup interval should be.

On most systems the backup interval is daily, but you can choose any other convenient interval. It is not necessary, though, to back up all the files in a file system at each backup. You can back up only those files that have changed since a previous backup. This is called an *incremental backup.* Using the **dump** and **restore** commands it is possible to back up to nine levels of incremental backups. For example, while a level 0 (zero) dump backs up an entire file system, a level 1 dump only backs up those files that have been created or modified since the last level 0 (zero) dump, and a level 7 dump only backs up those files created or modified since the last lower-level dump.

To integrate incremental backups into your file backup schedule, you need to balance the time and tape space required to make a backup against the amount of time it could take you to restore the system in the event of a system failure. For example, a possible backup schedule is to make backup levels following the ten day sequence:

0 1 2 3 4 5 6 7 8 9

On the first day you save an entire file system (level 0 [zero]). On the second day you save changes since the first backup and so on until the eleventh day when you restart the sequence. This makes the amount of time spent and data saved on each backup relatively small each day except the first one, but should a system failure require you restore the entire system on the tenth day, you must restore all ten tapes to the system.

Most systems follow some variant of the common *Tower of Hanoi* backup schedule. Once a month you make a level 0 (zero) dump to tape of all the regularly backed-up file systems. Then once a week you make a level 1 dump to start a daily sequence of:

3 2 5 4 7 6 9 8 9 9 ...

If you only do backups once a day on the weekdays, you end up with a monthly release schedule as follows.

0 1 3 2 5 4 1 3 2 5 4 ...

This schedule, although slightly complex to manage, requires you to restore at most four tapes at any point in the month should there be a system failure that corrupts files. Of course, doing a level 0 (zero) dump daily requires you to restore at most one tape at any point, but requires a large amount of time and tape storage to make each backup, where most days in the Tower of Hanoi schedule require very little time and tape storage for a backup.

Performing a Full Backup

At some point you should perform a full backup of each file system on your entire system including all the system software. You should set up a schedule for performing this backup just as for other backups. A conservative schedule for full system backups is to do one with each normal level zero dump (using Tower of Hanoi, once a month), but you can set any schedule you like within the reliability of your storage media, which is about two years for magnetic tapes.

The command used to perform file system backup is **dump**. It has the following syntax:

dump [*options filesystem*]

Where *options* is a list of flags and their arguments, and *filesystem* is the file system to be backed up. You should specify the file system as a full pathname. The **dump** command can only back up a single file system at a time. But there may be several **dump** processes writing files to selected tape devices.

The manual page for the **dump** command describes its options, which are mostly used to specify the characteristics of the tape device, such as block size, tape storage density, and tape length. Refer to the manual page for more information about the **dump** command options. The following text describes the most commonly used options to the **dump** command.

-integer Specifies the dump level as an integer (0-9). A dump level of 0 causes a full dump of the specified file system. All other dump levels cause an incremental backup. That is, only files that have changed since the last dump of a lower dump level are backed up. The file */etc/dumpdates* contains a record of when the **dump** command was used on each file system at each dump level. The *dumpdates* file is updated by the **-u** option to **dump**.

-f*dump_file* Writes the dump to the device specified by *dump_file* instead of the default device, */dev/rmt0h*. When *dump_file* is specified as - (dash), **dump** writes to the standard output.

-u Updates the file */etc/dumpdates* with the time of the dump and the dump level for the file system in the backup. This file is used during incremental dumps (by using the dump level option), to determine which files have changed since a particular dump level. */etc/dumpdates* is an ASCII file that you may edit to change any record or fields if it is necessary. The manual page for **dump** describes the format of this file.

To perform a backup of your entire file system to the default backup device, enter the following command for each file system on your machine, using a different tape device to store each file system:

```
dump -0u filesystem
```

Where *filesystem* is the name of a file system on your machine. The **-0u** option make a level 0 (zero) dump and updates the file */etc/dumpdates* with the time and date of the backup for each file system.

To perform a level 0 (zero) dump of the / (*root*), */usr*, and */projects* file system partitions, first enter the command:

```
dump -0u /
```

When the *root* file system is completely backed up, change to another tape on the tape drive and then back up the next (*/usr*) file system with the following command.

```
dump -0u /usr
```

When the */usr* file system is completely backed up, change the tape once again and finish the backup with the following command.

```
dump -0u /projects
```

Updating the */etc/dumpdates* file with the **-u** option creates an initial point on which to base all future incremental backups until the next full, or level 0 (zero) dump. Note that each file system must be backed up individually. It is common to back up each file system on an individual tape, but it is also possible to back up multiple file systems on one tape (assuming that the tape drive in question does not have an end-of-file rewind feature). Also note that it is possible to set up different backup schedules for each file system.

Performing an Incremental Backup

You should set up a routine as part of your backup schedule so that it is easier to remember which backup to do on each day. That routine should include a mechanism for logging your backups, their dump level, and listing the tapes they are made on. This information should not be on the computer system, because when you most need it the system is corrupted.

Once you have a system in place for making incremental backups, the procedure is quite simple. For example, assume you do a daily backup of */usr* using the following backup schedule.

```
0 1 9 9 9 1 9 9 9 9 ...
```

On the first Monday of the month you make a level 0 (zero) dump as shown in the previous section. On Tuesday you make a level 1 dump using the following command:

```
dump -1u /usr
```

The level 1 dump backs up all the files that have changed since Monday. On Wednesday through Friday you make a level 9 dump, which always backs up all the files that have changed since Tuesday's level 1 dump, by entering the following command:

```
dump -9u /usr
```

To make the same level 9 dump to the tape device referenced */dev/rmt1h* instead of the default tape device, you should use the **-f** option to dump as shown in the following command line:

```
dump -9u -f /dev/rmt1h /usr
```

The tape device specified by the argument to the **-f** option must be a device local to the system from which you are making the dumps.

Performing Remote Backups

Not every machine in a networked system has an ideal tape drive for making backup tapes. You can use the **rdump** command to make backups on a remotely located tape device. The **rdump** command is nearly identical to the **dump** command except that it requires the **-f** option to specify the machine name and an attached backup device. The **rdump** command has the following syntax:

rdump -f *machine:device* [*options filesystem*]

Where *machine* is the name of the remote machine with the backup device and *device* is the name of the backup device on that remote machine. The colon (:) between the machine and device parameters is necessary just as in other network file addressing mechanisms. The *options* parameter refers to the same list of flags available with the **dump** command. The *filesystem* parameter refers to the local file system to be backed up.

The manual page for the **dump** command describes all of the options to the **rdump** command.

To back up the */project* file system from the machine named *machine1* onto a tape drive on the machine named *machine2* having the attached backup device */dev/rmt0h*, enter the following command from *machine1*:

```
rdump -f machine2:/dev/rmt0h -0u /project
```

The **rdump** command updates the */etc/dumpdates* file on the local machine in the same way as the **dump** command does.

The **rdump** command starts a remote server, */usr/sbin/rmt*, on the remote machine to access the storage medium. This server process should be transparent. See the manual page for **rmt** for more information about this command.

Backup Scripts

To make the backup process less of a daily chore, you should automate it using shell scripts. These shell scripts can then be automatically executed by the **cron** daemon late in the evening when there is less chance of the **dump** commands making errors due to a changing system.

Backup shell scripts often include the following features:

- Determining the dump level
- Warning the system of the dump
- Making a listing of tape contents
- Notifying the operator upon completion

Some time during the day you simply install a tape reel into the tape drive, and at the specified time, the cron daemon runs the backup shell scripts. When the shell procedures are finished you remove the backup tape and archive it. Note that backup shell scripts are best used when the dump is small enough to fit on a single tape. When a dump does not fit on a single tape, someone must be available to physically change the tape when required.

Restoring Data

It is very optimistic to think you will never have to retrieve files from your backup tapes. You will likely even need to restore entire file systems at some time. If you have set up a good backup procedure then restoring files or even full file systems should be a simple task. This section describes how to restore your system, full file systems, or just a few files.

Restoring Your Entire System

When restoring your entire file system becomes necessary, a serious problem has occurred. It is usually not enough to simply restore the files and continue working. You should first determine what caused the problem, or you may wind up repeating the restore process. The most important step is re-installing your system from the initial boot tapes. The installation instructions that came with your system explains this procedure.

Once you have a minimum system up and running, you need to restore that system to the state it was in just prior to the system crash. You can use the **restore** command to restore data from tapes made with the **dump** command. Because the **dump** command only saves a single file system at a time, you will need to execute the **restore** command for each file system you wish to restore.

The restore command has the following syntax:

```
restore options
```

Where *options* is a list of flags and their arguments. The manual page for the **restore** command describes all its options, which are mostly used to specify the characteristics of the tape device and special restore options. Refer to the manual page for more information about the options to the **restore** command. The following text describes the most commonly used options to the restore command.

-i Starts interactive restoration of files from the tape. After reading directory information from the tape, this option provides a shell-like interface that allows you select the files you want to restore. The commands available in interactive mode are described under "Restoring Files Interactively" on page 7-10.

-r Restores the entire contents of the file system on the backup tape into the current working directory. This should usually not be done except to restore an entire file system into an empty directory or to restore file system incremental dumps.

-t *names* Creates a listing of files and directories on the tape that match *names*. If *names* is specified, **restore** returns a list of the filenames and directories that are on the tape that match the names listed. The names should be specified as *./filename*. If *names* is not specified, restore returns a complete listing of the backed up files on the tape.

-x *names* Restores the files and directories from the tape specified by *names*. The *names* argument contains a list of files and directories to be restored from the tape. The names should be specified as *./filename*. If *names* specifies a directory name, then all the files in the directory are recursively restored.

-f *dump_file* Restores the dump from the device specified by *dump_file* instead of the default device, */dev/rmt0h*.

-F *file* Specifies a file from which interactive restore commands are read. This option should be used in conjunction with the **-i** option.

Restoring the entire system at this point is a matter of building the new file systems and restoring the dumps from the tapes.

Start off by restoring the contents of the / (*root*) file system because that file system exists initially on every system. Restore the / file system by installing its most recent level 0 (zero) tape reel and entering the following commands.

```
cd /
restore -r
```

Change tapes as needed until the tape is restored. When that tape is fully restored, follow the same procedure for each incremental tape in your dump sequence until the entire *root* file system is restored. Install the incremental dump tape reel and enter the following command for each dump level.

```
restore -r
```

Once the *root* file system is completely restored, you need to mount each new file system and restore its contents. The procedure for restoring each of the file systems in turn is similar to installing the *root* file system and is described in the following section.

Restoring a Full File System

Restoring a full file system requires that you first make a place to restore the file system to and then use the **restore** command to restore the entire contents of the backup tapes for the file system. You make a place for the file system using the **newfs** and **mount** commands as shown in the following example.

```
newfs raw_device disk_type
mount block_device filesystem
cd filesystem
```

Where *raw_device* is the full raw device pathname of the disk device on your system, *block_device* is the full block device pathname of the disk device on your system, *disk_type* is the type of the disk device associated with *raw_device* as defined in the file */etc/disktab*, and *filesystem* is the full pathname of the file system you want to create. Creating and mounting a file system is described in detail in Chapter 10.

Once the empty file system is created use the **restore** command with the **-r** flag to restore the most recent level 0 (zero) dump and the subsequent incremental dumps into the newly recreated file system. Install each backup tape reel in turn and enter the following **restore** command:

```
restore -r
```

Remember that this command must be issued from within the file system that you are restoring.

Restoring Files

It is far more common that you will need to restore a single lost file than an entire file system. When users lose files they must ask their system administrator to have those files restored. Users may also ask that you restore an earlier version of a file. Whatever the reason for a file restoration, the procedure is the same: find out which tape has the correct version of the file and restore that file using the restore command.

By asking when the file was lost and when it was last modified before it was lost, you can use your backup log to make a good guess as to which tape has the most recent version of the wanted file. You can then use the **-t** option with the **restore** command to determine whether a file is on the selected tape using the following form of the command:

```
restore -t names
```

The **-t** option creates a listing of files and directories on the tape that match names. The names should be specified as *./filename* for each name. For example, to see the contents of the *working* subdirectory of the */usr* file system on a particular backup tape, install the tape reel and issue the following command.

```
restore -t ./working
```

To make a listing of the entire contents of a backup tape, install the backup tape reel and issue the following command.

```
restore -t
```

It is a good idea to make a listing of every backup tape after you create it. This gives you a place to quickly look up what files are on the tape and also verifies a successful backup.

Once you have determined on which tape the file you want to restore is located, you should restore it using the following form of the restore command.

```
restore -x names
```

Where the **-x** option is specified in the same way as the **-t** option. The files are restored into your current working directory. You should be careful not to restore files in a place where they might overwrite existing files. For that reason it is usually best to restore files into a newly created directory.

To restore the file *working/old.file* from a */usr* file system backup tape into your current directory, install the backup tape reel and issue the following command.

```
restore -x ./working/old.file
```

To restore the entire contents of the working subdirectory from the same tape issue the following command.

```
restore -x ./working
```

Restoring Files Interactively

The **-i** option to the restore command lets you start an interactive restore session. This makes restoring a number of files much easier. The interactive mode has commands similar to shell commands that let you list the contents of a tape, move around the directory hierarchy of the tape, add files to restore, and delete files to restore among other things. You enter interactive restore mode by entering the following command.

```
restore -i
```

Once in the interactive restore mode there are a number of commands available, which are described in the following list:

ls [**-v**] [*directory*]
: Lists files in the current or specified directory. Directory entries are appended with a / (slash) character. Entries that have been marked for reading are prepended with a * (asterisk) character. When the **-v** modifier flag is used, the inode number of each entry is also listed.

cd [*directory*]
: Changes the current directory to the directory specified by *directory*.

pwd
: Lists the pathname of the current directory.

add [*files*]
: Adds the files in the current directory or the files specified by *files* to the list of files read from the tape. Files are marked with the character * (asterisk) when they are listed with the **ls** interactive command once they are specified to be read by this interactive command.

delete [*files*]
: Deletes all the files in the current directory or the files specified by *files* from the list of files read from the tape.

extract
: Actually restores the files from the tape that are marked to be read into the current working directory. The **extract** command prompts you for the logical volume that you want to mount (usually 1), and whether the access modes of . (dot) are affected (answer yes when you are restoring the entire root directory).

setmodes
: Sets owner, access modes, and file creation times for all directories that have been added to the files-to-read list; nothing is read from the tape. This interactive command is useful for cleaning up files after a **restore** command has been prematurely aborted.

verbose
: Toggles verbose mode. In verbose mode, as each file is extracted its name is printed to the standard output. The default state of verbose mode is off. This is the same as the **-v** command line option to **restore**.

help
: Lists a summary of these interactive commands.

?	Lists a summary of these interactive commands.
what	Lists the tape header information.
quit	Quits the interactive restore session.
debug	Toggles the debugging mode.
xit	Quits the interactive restore session.

To interactively restore the two files *./working/file1* and *./working/file2* from a backup tape, install the tape reel and enter the command:

```
restore -i
```

Once in interactive mode you enter the following commands to add the files to the list of files to be extracted:

```
cd working
add file1
add file2
```

You then extract the files with the following command:

```
extract
```

You are then prompted for the logical volume you wish to mount. You usually respond to this prompt with **1**. Then you are asked whether the extract affects the access modes of the **.** (dot). For this example, the reply is **no**.

Once the files are extracted you can quit the interactive restore mode by entering the following command:

```
quit
```

The files *file1* and *file2* should then be found in the current directory.

All these steps can be automated in a *restore script* that is read by the **-F** option to **restore**. To do this, first create a restore script for the files you want to restore. For example, the following restore script in the file named *restore_file* performs the restore operation requested in the previous example:

```
cd working
add file1
add file2
extract 1
no
quit
```

This restore script is read and executed by the following command:

```
restore -iF restore_file
```

The results of this restore script procedure are identical to our previous interactive restore session.

Performing Remote Restores

Just as you may need to make remote backup tapes, you may need to perform remote restores. You can use the **rrestore** command to perform restores to local directories from a remote tape device. The **rrestore** command is nearly identical to the **restore** command except that it requires the **-f** option to specify the machine name and its backup device. It has the same syntax as the **-f** option to the **rdump** command.

```
rrestore -f machine:device [options]
```

Where *machine* is the name of the remote machine where the backup device is attached, and *device* is the name of the backup device on that remote machine. The **:** (colon) between *machine* and *device* is necessary just as in other network file addressing mechanisms. The *options* parameter is the same list of flags available for the **restore** command.

The manual page for the **restore** command describes all of the options to the **rrestore** command.

To restore the file *./working/file1* onto the local directory on the machine named *machine1* from a backup tape mounted on *machine2* where the backup device */dev/rmt0h* is attached, enter the following command from *machine1*:

```
rrestore -f machine2:/dev/rmt0h -x ./working/file1
```

The **rrestore** command starts a remote server, */usr/sbin/rmt*, on the remote machine to access the storage medium. This process should be transparent. See the manual page for **rmt** for more information about this command.

Monitoring and Operating DAT Drives

You can monitor the activity of each drive through two front panel display Light Emitting Diodes (LEDs). One LED is dedicated to Cassette activity, the other is dedicated to SCSI drive activity. Physically, the cassette light is just above the drive light in the upper left hand side of the drive unit. Each LED can display two colors:

Green	Indicates normal operation.
Amber	Indicates warning conditions.

Monitoring Cassette and Drive Activity

The hardware displays the LEDs in either the ON, OFF, or PULSING states depending on the hardware activity. Table 7-1 describes all options.

Table 7-1. Front Panel Display

Status	Cassette Display	Drive Display	Description
Read/Write States	Pulse Green	Pulse Green	Cassette (Un)Loading
	Green	Green	Cassette Loaded/Online
	Green	Pulse Green	Cassette Loaded/Activity
	Green	Off	Cassette Loaded/Offline
Write Protect States	Pulse Amber	Pulse Green	Cassette (Un)Loading
	Amber	Green	Cassette Loaded/Online
	Amber	Pulse Green	Cassette Loaded/Activity
	Amber	Off	Cassette Loaded/Offline
Error States	Green	Alternating Green/Amber	Media Wear (caution)
	Amber	Amber	High Humidity/No termination
	Pulse Amber	Pulse Amber	Self-test (normal)
	Pulse Amber	Amber	Self-test (failure)

Monitoring System Activity

The front panel also displays unique system level activity:

Caution

Indicates an error correction signal or an excessive number of read-after-write operations because of corroded or dirty cassette heads (or possibly a faulty tape).

Caution Indication

Cassette light is steady green and the drive light alternates green and amber.

Action

Clean the tape drive heads and try the operation again. If the caution signal reappears, copy the data from the cassette onto a new one and remove the old cassette. To reset the caution signal indication, merely unload the cassette.

High Humidity

Indicates high humidity condition in the drive unit. A steady amber condition may also indicate that no terminating resistor exists on the SCSI bus.

High Humidity Indication
: Both the cassette light and the drive light are steady amber.

Action
: If high humidity is the problem, you must conform to the environmental specifications for humidity. If the display is caused by no termination, call the manufacturer for proper termination procedures on the SCSI bus.

Diagnostic Error

Indicates an diagnostic error detected during power up. During the power-up tests, the display lights flash amber at a 2 Hertz rate.

Diagnostic error indication
: The drive light changes to steady amber.

Action
: Rerun the diagnostic test (power down, then power-up). If the problem persists, call service or replace the unit.

NOTE

The host can also initiate diagnostic tests of the drive. The results of these tests are reported to the host.

Operating Hints

Conforming to the following procedures will prevent operating errors:

Loading a Cassette

The loading procedure takes about 25 seconds as the drive rewinds to the beginning of tape (BOT) and goes online.

- Use only certified cassettes (visible with a DDS sticker).
- Always load the cassette with the label at the top and with the arrows pointing toward the drive unit.
- Do not force the cassette into the drive. Allow the autoloading mechanism to operate.

Unloading a Cassette

- Press the unload button to eject the cassette.The tape automatically rewinds to the BOT and, if write-enabled, a copy of the tape log held in RAM is written back to the tape.

Cleaning heads

You should clean the heads after every 25 hours of use or when the LEDs display the caution signal.

- Insert a special cleaning cassette into the drive. The drive automatically loads the cassette and cleans the head. At the end of the cleaning cycle, the drive automatically ejects the cassette.
- Discard the cleaning cassette after 25 uses.

Write-protecting cassettes

- Slide the tab on the rear of the cassette so that the hole is visible. Note that a tape log (tape usage history) cannot be updated when the cassette is write-protected.

Cleaning the 4mm DAT Tape Drive

Your cartridge tape drives should be cleaned periodically. You can obtain a cleaning cartridge either from your media supplier or from Hewlett-Packard. Use the HP DAT Cleaning Cassette, HP 92283K (or equivalent) as follows:

Insert the cleaning cassette into the drive. The drive will automatically load the cassette and clean the heads. At the end of the cleaning cycle, the drive will automatically eject the cleaning cassette.

Please note the date of the cleaning on the cleaning cassette label. The cleaning cassette can be used up to 25 times, after which it should be discarded.

Configuring Partitions 8

Introduction

The nodes of the Paragon™ system are grouped into *partitions*. Partitions control the way the nodes are used; for example, users' shells and other non-parallel processes run in the *service partition*, while parallel applications run in the *compute partition*. The nodes of the compute partition can be grouped into *subpartitions* to provide finer control.

NOTE

This chapter discusses partitions of computing nodes, not disk partitions.

The system administrator is responsible for managing the system's partitions. When you are logged in as *root*, you have the same degree of control over partitions that you do over files and directories (and you must exercise the same degree of care with this control).

This chapter explains what partitions are and how you manage them. It includes the following sections:

- Partition concepts.
- Partition management software.
- Guidelines for configuring partitions.
- Creating a new partition configuration.

Partition Concepts

This section discusses a few partition concepts that all Paragon system administrators must know. For complete information about partitions, see the *Paragon™ System User's Guide.*

Partition Basics

A partition is a group of nodes with an associated set of attributes that control how applications run within the partition. Each partition has the following attributes:

- A parent partition.
- An owner and group.
- A set of protection modes.
- A set of scheduling characteristics.

A partition and its attributes are persistent across reboots. A partition's attributes define how the partition can be used; for example, a partition's protection modes determine who can execute applications in the partition and who can create subpartitions within the partition.

Partition Hierarchy and Special Partitions

Partitions are arranged in a hierarchical structure, like the hierarchical directory structure of the UNIX file system. The nodes of each partition are a strict subset of the nodes of its parent partition; that is, each child partition is the same size as its parent or smaller, and no partition ever contains a node that is not part of its parent. The top partition in this hierarchy is called the *root partition*; it contains every usable node in the system.

Immediately below the root partition, all Paragon systems must have two special partitions called *service* and *compute*. The *service* partition is the partition in which users' shells and other commands run; the *compute* partition is the partition in which parallel applications run. You can also choose to configure your system with an *I/O partition*: a third special partition that contains the nodes that control disks, tapes, and other I/O devices. This partition is usually called *io*.

In most Paragon systems, the only partitions in the system other than the root, service, compute, and I/O partitions are subpartitions of the compute partition. These partitions may be created by the system administrator or by ordinary users, depending on the permissions of the compute partition and the policies of your site. The characteristics of these partitions are determined by the user who creates them.

Partition Pathnames

Like directories, partitions are identified by pathnames. A *partition pathname* is like a directory pathname, except that the parts of the pathname are separated by periods (.) rather than slashes (/). The root partition's pathname is . (dot), the service partition's pathname is *.service*, and the compute partition's pathname is *.compute*. The pathname of the I/O partition is usually *.io*.

If *mypart* is a subpartition of the compute partition, its pathname is *.compute.mypart*. A subpartition of the compute partition can also be identified by a *relative partition pathname*, which consists of its full pathname with the *.compute* stripped off the beginning. For example, the relative partition pathname of *.compute.mypart* would be just *mypart*. (Relative partition pathnames are *always* relative to the compute partition.)

Node Numbers in Partitions

Each node in a partition has a *node number* within the partition: an integer from 0 to one less than the partition's size. The nodes in a partition are typically numbered from left to right and then from top to bottom. However, if a partition was created with an explicit node list (using the **nx_mkpart_map**() call or the **-nd** switch of the **mkpart** command), the nodes are numbered in the order they were specified in that list.

Note that, because partitions can overlap, a single node can have more than one node number. For example, if a partition has four nodes (numbered 0, 1, 2, and 3) and it has a subpartition that consists of nodes 2 and 3, node 2 of the partition is also called node 0 of the subpartition.

You can uniquely identify the physical position of a node within the system by giving the node number of that node in the root partition. This number, called the *root node number* or *OS node number*, does not change unless the system is substantially reconfigured (such as adding or removing a cabinet).

Physical nodes can also be identified by a *CBS* (Cabinet, Backplane, Slot) number; see "CBS Numbering" on page 1-10 for details on the CBS numbering system.

Application Scheduling in Partitions

When a parallel application is run, it runs in the compute partition or a subpartition of the compute partition. The user can specify the partition when the application is invoked; if the user does not specify otherwise, the application runs in the compute partition. An application cannot use any nodes other than the nodes of its partition, so the partition places a limit on the size of the application. However, an application doesn't have to run on all the nodes of the partition.

A partition that contains a running application is called an *active* partition. Note that if an application is running *anywhere* in a partition or any of its subpartitions—even on a single node—the *entire* partition is considered active.

If an application's processes are the only processes on the specified nodes, the application runs until it completes. However, when two or more active entities (running applications or active partitions) overlap, the *scheduling type* of the partition that contains them determines which ones run and for how long. A partition's scheduling type is one of the following:

- In *standard scheduling*, overlapping entities run at the same time; the processes on each node compete for CPU time using the standard UNIX scheduling policies. This can deliver poor performance for message-passing applications, because there is no guarantee that all the processes of an application are running at the same time.

- In *space sharing*, overlapping entities are not allowed. Any attempt to run an application that would cause applications and/or active partitions to overlap fails immediately with the error "request overlaps with nodes in use."

- In *gang scheduling*, overlapping entities are scheduled so that only one entity is active on a node at one time, and an entire entity is active at once across all the nodes on which it is loaded. The granularity of scheduling is controlled by the partition's *rollin quantum*, a length of time up to 24 hours.

  If you set the rollin quantum to 0, once an application is rolled in it is not rolled out until it completes (unless the partition itself is rolled out during that time). Applications that would overlap with a running application are queued for later execution. This special case of gang scheduling is sometimes called *FIFO scheduling*.

  Gang scheduling offers the maximum flexibility for running large applications, but it can stress the system severely and may decrease system stability. As the system administrator, you can configure the system to prevent or restrict the use of gang scheduling.

The root partition usually uses space sharing, the service partition always uses standard scheduling, the compute partition usually uses either space sharing or gang scheduling, and subpartitions of the compute partition usually use space sharing. See the *Paragon™ System User's Guide* for more information on application scheduling in partitions.

Subpartitions in Partitions

Whether or not a partition can have subpartitions is determined by its scheduling type:

- A partition that uses standard scheduling cannot have subpartitions.

- A partition that uses space sharing can have subpartitions, but they may not overlap each other.

- A partition that uses gang scheduling can have subpartitions, and the subpartitions can overlap (up to a maximum depth that is set by the system administrator).

Note that a subpartition does not have to be the same type as its parent. For example, a space-shared partition can have gang-scheduled subpartitions. However, if the parent of a space-shared partition is gang-scheduled, the benefits of space sharing may be lost.

Partition Management Software

This section introduces the software you can use to manage partitions on your system. Partitions are managed on the Paragon system itself; all the files, commands, and calls described in this section are found on the Paragon system, not the diagnostic station.

Partition Management Commands And Calls

Most of the commands and calls used by the system administrator to manage partitions are the same as those used by ordinary users. However, only *root* can use these commands and calls to perform the following partition management tasks:

- Change the attributes of the root, service, I/O, and compute partitions.
- Change the ownership of a partition.
- Change the group of a partition to a group you do not belong to.
- Change the attributes of a partition you do not own.
- Perform any action on a partition (create or remove subpartitions, get information on the partition, or run applications in the partition) when its permissions do not allow you to do this.

For example, to change the ownership of the partition *interactive* to user *chris* and group *users*, use the following command:

```
# chpart -o chris.users interactive
```

The partition management commands are summarized in Table 8-1; the partition management calls are summarized in Table 8-2. See the *Paragon™ System User's Guide* for more information on these commands and calls.

The commands and calls in Table 8-1 and Table 8-2 are available to all users. The rest of this section discusses partition management software that is only available to the system administrator.

Partition Configuration Files

The current state of the system's partitions is recorded in a directory structure whose root directory is */etc/nx* on the Paragon system. For example, each subpartition of the root partition is represented by a directory in */etc/nx*, and each subpartition of the compute partition is represented by a directory in */etc/nx/compute*. These directories and the files within them are created and maintained by the **initpart** command and the allocator daemon (described later in this section).

Table 8-1. Partition Management Commands

Command Synopsis	Description
mkpart [**-sz** *size* | **-sz** *h*X*w* | **-nd** *nodespec*] [**-nt** *nodetype*] [**-ss** | [[**-sps** | **-rq** *time*] [**-rlx**] [**-epl** *priority*]]] [**-mod** *mode*] *partition*	Create a partition.
rmpart [**-f**] [**-r**]*partition*	Remove a partition.
showpart [**-f**] [*partition*][**-p** | **-l**] [**-w**][**-nt** *nodetype*]	Show the characteristics of a partition.
lspart [**-r**] [**-w**] [**-p** | **-l**] [**-r**] [**-nt** *nodetype*] [*partition*]	List the subpartitions of a partition.
pspart [**-r**] [*partition*]	List the applications in a partition.
chpart [**-epl** *priority*] [**-g** *group*] [**-mod** *mode*] [**-nm** *name*] [**-o** *owner*[**.** *group*]] [**-rq** *time* | **-sps**] *partition*	Change certain partition characteristics.

The characteristics of each partition are recorded in a file called *.partinfo* in the */etc/nx* directory corresponding to that partition. For example, the characteristics of the root partition are recorded in the file */etc/nx/.partinfo*, and the characteristics of the compute partition are recorded in the file */etc/nx/compute/.partinfo*.

NOTE

The format of *.partinfo* files is not documented and is subject to change without notice.

Never write programs that use the information in these files to determine the current state of the system's partitions, never edit these files, and never modify or remove these files or their directories while the allocator daemon is running! Always use standard commands and calls such as **chpart** and **nx_part_nodes()** to change partitions and get information about partitions. However, you need to know that these files exist; for example, you need to back up everything under */etc/nx* regularly so that you can restore your partition configuration from backups when necessary.

Table 8-2. Partition Management Calls

Call Synopsis	Description
nx_mkpart(*partition, size, type*)	Create a partition with a particular number of nodes.
nx_mkpart_rect(*partition, rows, cols, type*)	Create a partition with a particular height and width.
nx_mkpart_map(*partition, numnodes, node_list, type*)	Create a partition with a specific set of nodes.
nx_rmpart(*partition, force, recursive*)	Remove a partition.
nx_part_attr(*partition, attributes*)	Get a partition's attributes.
nx_part_nodes(*partition, node_list, list_size*)	List the root node numbers for the nodes of a partition.
nx_pspart(*partition, pspart_list, list_size*)	Obtain information about all applications and active subpartitions in a partition.
nx_chpart_name(*partition, name*)	Change a partition's name.
nx_chpart_mod(*partition, mode*)	Change a partition's protection modes.
nx_chpart_epl(*partition, priority*)	Change a partition's effective priority limit.
nx_chpart_rq(*partition, rollin_quantum*)	Change a partition's rollin quantum.
nx_chpart_owner(*partition, owner, group*)	Change a partition's owner and group.
nx_chpart_sched(*partition, sched_type*)	Change a partition's scheduling type.
nx_empty_nodes(*node_list, list_size*)	List the nodes that are empty slots.
nx_failed_nodes(*node_list, list_size*)	List the nodes that failed to boot.

Service Partition Initialization at Boot Time

When the system boots, the operating system reads the file */etc/nx/service/.partinfo* to determine which nodes are in the service partition. If this file does not exist, the operating system assumes that the service partition consists of only the boot node. This information is used by the operating system and the load leveler daemon to distribute processes among the nodes of the service partition (see **load_leveld** in the *Paragon™ System Commands Reference Manual* for more information on the load leveler daemon).

Root Partition Initialization at Boot Time

The **initpart** command is run automatically when the system is coming up to a multiuser state (state 3 or higher). **initpart** makes sure that the root partition accurately describes the complete set of usable nodes in the system, as described by the bootmagic strings *BOOT_MESH_X*, *BOOT_MESH_Y*, and *BOOT_NODE_LIST*:

- If the file */etc/nx/.partinfo* does not exist, or if the root partition width and height in that file do not match *BOOT_MESH_X* and *BOOT_MESH_Y*, **initpart** creates a new root partition, as follows:
  - Removes all partition directories under */etc/nx* (thus removing all partitions). This is necessary because all partitions are based on the root partition; if the root partition is invalid, all partitions must be considered invalid.
  - Creates a new */etc/nx/.partinfo* file with the width and height specified by *BOOT_MESH_X* and *BOOT_MESH_Y*. Any nodes within this rectangle that do not appear in *BOOT_NODE_LIST* are marked as unusable nodes. All other characteristics of the new root partition are set to default values.
  - Creates a new */etc/nx/allocator.config* file with the factory default values for all parameters. See "The Allocator Configuration File" on page 8-10 for information on this file.
- If the root partition has the correct width and height but the root partition node list in */etc/nx/.partinfo* does not match *BOOT_NODE_LIST*, **initpart** updates the node list in */etc/nx/.partinfo* and does nothing else.
- If the root partition matches *BOOT_MESH_X*, *BOOT_MESH_Y*, and *BOOT_NODE_LIST*, **initpart** does nothing.

See the **initpart** manpage in the *Paragon™ System Commands Reference Manual* for more information on this command.

The Allocator

After **initpart** has run, the *allocator* daemon is started. This daemon allocates nodes to partitions and applications; handles requests to create, modify, or remove partitions; and manages the scheduling of applications in space-shared and gang-scheduled partitions. (The allocator does not manage processes within standard-scheduled partitions.)

The allocator reads the partition configuration files under */etc/nx* when it starts up, to determine its initial partition configuration. The allocator then maintains an image of the current partition configuration in its memory, and updates the partition configuration files whenever this configuration changes. (See "Partition Configuration Files" on page 8-5 for more information on these files.) Note that after the allocator has started up, it does not read these files; it only writes them.

For example, when you create a partition with the **mkpart** command, the command sends a request to the allocator for the specified nodes. If the request cannot be fulfilled (because, for example, the specified parent partition does not exist, or the parent partition uses space sharing and some of the requested nodes already belong to subpartitions), the allocator rejects the request and the **mkpart** command fails. Otherwise, the allocator updates its internal data structures, then makes a directory for the new partition under */etc/nx* and creates a *.partinfo* file in that directory. For those cases where you want as many nodes as possible, even if some nodes are not available, use the **-rlx** switches in the **mkpart** command. See the **mkpart** manpages for more information.

Similarly, when you run an application, the application sends a request to the allocator for the specified nodes. If the request cannot be fulfilled (because, for example, you do not have execute permission for the specified partition, or the partition uses space sharing and some of the requested nodes are already in use by other applications or active partitions), the allocator rejects the request and the application does not run. Otherwise, the allocator allocates the requested nodes to the application. (This allocation is exclusive if the partition uses space sharing and nonexclusive otherwise.) Once the allocator has allocated nodes to an application, the way the application runs is determined by the partition's scheduling type:

- In a standard-scheduled partition, the application's processes use standard UNIX scheduling mechanisms to compete with any other processes on their nodes for processor time. The allocator does nothing once the application starts.
- In a space-shared partition, the application retains exclusive use of its nodes until it terminates. The allocator rejects any other requests to use those nodes until the application has finished using them.
- In a gang-scheduled partition, the application rolls in and out according to its priority. The allocator examines the partition at the end of each rollin quantum and whenever an application starts or finishes in the partition, and rolls applications in or out according to their priorities at that time.

See the *Paragon™ System User's Guide* for more information on partition permissions and scheduling. See the **allocator** manpage in the *Paragon™ System Commands Reference Manual* for more information on the allocator.

The Allocator Configuration File

The file */etc/nx/allocator.config* controls the behavior of the allocator daemon. By editing this file, you can perform the following tasks:

- Prevent or enable the use of gang scheduling.
- Specify the maximum number of gang-scheduled partitions.
- Specify the maximum degree of overlap in a gang-scheduled partition.
- Specify the minimum rollin quantum in a gang-scheduled partition.
- Prevent the use of **-plk** in a gang-scheduled partition.
- Specify whether or not your site is using MACS.

NOTE

The allocator reads the */etc/nx/allocator.config* file only when it starts up.

If you make any changes in this file, you must stop and restart the allocator before the changes take effect. (See "Other Files Used by the Allocator" on page 8-12 for information on how to do this.)

The */etc/nx/allocator.config* file is a text file that contains parameter definitions (lines of the form *parameter=value*) and comments (lines beginning with #). The following parameters are defined:

`SPACE_SHARE=`*boolean*

This parameter determines whether space sharing is enforced. The value of *boolean* must be 1 (no gang-scheduled partitions are allowed anywhere in the system) or 0 (gang-scheduled partitions are allowed).

The factory default for *SPACE_SHARE* is 0; if this line is omitted or commented out or the *allocator.config* file is missing, the default value is 0.

The *SPACE_SHARE* parameter is equivalent to the allocator **-tile** switch used in previous releases, and should be used instead of **-tile**.

NUM_GANG_PARTS=*integer*

This parameter determines the maximum total number of gang-scheduled partitions allowed anywhere in the system. The value of *integer* must be an integer greater than 0.

The factory default for *NUM_GANG_PARTS* is 1; if this line is omitted or commented out or the *allocator.config* file is missing, the default value is unlimited. (Note that there is no value of *integer* that specifies "unlimited;" the only way to remove the limit is to remove or comment out the *NUM_GANG_PARTS* line.)

DEGREE_OF_OVERLAP=*integer*

This parameter determines the maximum permissible depth to which subpartitions or active entities (running applications or active subpartitions) can overlap in a gang-scheduled partition. For example, if *integer* is 2, a single node can be allocated to at most two subpartitions and two active entities at the same time. The value of *integer* must be an integer greater than 0.

The factory default for *DEGREE_OF_OVERLAP* is 2; if this line is omitted or commented out or the *allocator.config* file is missing, the default value is unlimited. (Note that there is no value of *integer* that specifies "unlimited;" the only way to remove the limit is to remove or comment out the *DEGREE_OF_OVERLAP* line.)

MIN_RQ_ALLOWED=*time*

This parameter determines the minimum permissible rollin quantum. The value of *time* must be a time specification as used for the **-rq** switch of the **mkpart** or **chpart** command: an integer representing a number of milliseconds, or an integer followed by the letter **s**, **m**, or **h** to represent a number of seconds, minutes or hours. The specified time value must be less than 24 hours. The value 0, meaning "infinite" rollin quantum, cannot be used.

The factory default for *MIN_RQ_ALLOWED* is **1h** (one hour); if this line is omitted or commented out or the *allocator.config* file is missing, the default value is 100 milliseconds. See "Determining the Minimum Rollin Quantum" on page 8-14 for information on determining the appropriate value of *MIN_RQ_ALLOWED* for your system.

REJECT_PLK=*boolean*

This parameter determines whether the application switch **-plk** can be used in a gang-scheduled partition. The **-plk** switch locks parts of each process into physical memory, which reduces paging and improves message-passing latency but can cause problems when the application is rolled out; see the *Paragon™ System User's Guide* for more information on this switch. The value of *boolean* must be 1 (**-plk** is not allowed in gang-scheduled partitions) or 0 (**-plk** is allowed).

The factory default for *REJECT_PLK* is 0; if this line is omitted or commented out or the *allocator.config* file is missing, the default value is 0.

USE_MACS=*boolean*

This parameter determines whether the allocator must validate users' accounts with the Paragon Multi-User Accounting and Control System (MACS). The value of *boolean* must be 1 (validate users' accounts with MACS) or 0 (do not validate users' accounts).

The factory default for *USE_MACS* is 0 (this line is not present in the factory default *allocator.config* file); if this line is omitted or commented out or the *allocator.config* file is missing, the default value is 0.

The *USE_MACS* parameter is equivalent to the allocator **-MACS** switch used in previous releases, and should be used instead of **-MACS**.

See the **allocator.config** manpage in the *Paragon™ System Commands Reference Manual* for more information on the *allocator.config* file.

Other Files Used by the Allocator

The following additional files are used by the allocator daemon:

- The partition configuration files under */etc/nx* are read by the allocator when it starts up, and written by the allocator whenever it creates or changes a partition. These files record the current state of the system's partitions; see "Partition Configuration Files" on page 8-5 for more information.

- The file */etc/nx/.badnodes* lists the root node numbers of nodes that failed to boot. This file is created by the **bootmesh** program when the system boots, and is read by the allocator when it starts up. The allocator will not start any applications or make any partitions using these nodes. These nodes are marked with an X in the output of the **showpart** command, and you can get a list of these nodes by calling **nx_failed_nodes**().

- The file */etc/nx/allocator.log* records any status messages produced by the allocator since the last time it was started. Each time the allocator is started, the current allocator log file is renamed to */etc/nx/allocator.log.last* and a new */etc/nx/allocator.log* file is created.

- The script */sbin/init.d/allocator* is used to start and stop the allocator. Use the command **/sbin/init.d/allocator stop** to terminate the allocator daemon; use the command **/sbin/init.d/allocator start** to restart it. Note that commands and calls that manipulate partitions and applications will not work while the allocator is stopped.

NOTE

Do not stop the allocator while applications are running.

See the following section for more information on stopping and starting the allocator.

- The allocator daemon itself is found in the binary file */usr/sbin/allocator*.

Stopping and Starting Daemons

When stopping and starting the allocator, keep in mind the following dependencies among the various system daemons and background processes:

- Applications depend on the allocator.
- The System Monitor Daemon (SMD) depends on the allocator.
- MACS and NQS depend on SMD.
- NQS depends on MACS (if so configured).

If you want to stop any daemon or process that is depended on by another daemon or process, you must stop them in the correct order, as follows:

1. Stop NQS before stopping MACS.
2. Stop NQS and MACS before stopping SMD.
3. Stop SMD before stopping the allocator.
4. Kill all applications (or wait for them to terminate) before stopping the allocator.

Similarly, if you restart these daemons by hand, you must start them in the correct order, as follows:

1. Start the allocator before starting SMD.
2. Start SMD before starting MACS.
3. Start MACS before starting NQS.

Determining the Minimum Rollin Quantum

The factory default value for the *MIN_RQ_ALLOWED* parameter in */etc/nx/allocator.config* is one hour. This may or may not be suitable for your system; this section will help you determine the appropriate value for your system. You can skip this section if you don't use gang scheduling.

To determine the best value of *MIN_RQ_ALLOWED* for your system, you need to keep in mind that rolling applications in and out may require virtual memory paging. Whenever an application is rolled out and a new application rolled in, if the two applications can't fit in the nodes' memory at the same time the old application must be paged out to disk and the new application paged in from disk. This paging takes time. The value of *MIN_RQ_ALLOWED* should be set so that in the worst case (all physical memory in the largest gang-scheduled partition needs to be paged both out and in), there is enough time left over for the application to do some reasonable amount of work before the rollin quantum ends.

The time required to page out the old application or page in the new application is approximated by the following formula:

$$T = (M * N) / (R * P)$$

where:

- T is the total time required to page the entire application out or in (seconds).
- M is the amount of memory being paged out or in per node (MB).
- N is the number of nodes in the application.
- R is the rate at which pages can be written to or read from disk by each paging node (MB/sec). This rate is approximately 0.5 MB/sec in the current release.
- P is the number of paging nodes being used in the operation.

For example, suppose that each node has 32 MB of physical memory ($M = 32$), the largest gang-scheduled partition has 72 nodes ($N = 72$), the paging rate is 0.5 MB/sec for both writes and reads ($R = 0.5$), and there are 8 paging nodes providing paging to those 72 nodes ($P = 8$). In this case, the largest possible application would take approximately ((32 * 72) / (0.5 * 8) = 576) seconds, or 9.6 minutes, to page out. If the application being paged in is equally large, it would also take 9.6 minutes to page in, so as much as 19.2 minutes of each rollin quantum can be spent in paging. This means that setting a partition's rollin quantum to less than 19.2 minutes could cause the system to hang, because it may try to page an application out when it is not completely paged in yet.

Once you have determined the worst-case time for paging, you need to set *MIN_RQ_ALLOWED* to a large enough value that the overhead from paging is reduced to a reasonable percentage of each rollin quantum. For example, if you set *MIN_RQ_ALLOWED* to one hour in this case, the maximum overhead from paging (19.2 minutes) would be approximately 30% per rollin quantum. A larger value of *MIN_RQ_ALLOWED* would reduce this overhead percentage, but would also reduce the system's gang-scheduling flexibility.

Layering in Gang-Scheduled Partitions

The factory default value for the *DEGREE_OF_OVERLAP* parameter in */etc/nx/allocator.config* is 2. This section explains some of the internals of how the allocator schedules gang-scheduled partitions to help you understand what this parameter means. You can skip this section if you don't use gang scheduling.

Internally, the allocator maintains a "layered" image of each active gang-scheduled partition. Each layer consists of a set of non-overlapping applications and/or active subpartitions. Whenever an application is run in a gang-scheduled partition or one of its subpartitions, the allocator attempts to fit the new application or active subpartition into one of the existing layers; if it cannot, it creates a new layer. The *DEGREE_OF_OVERLAP* parameter determines the maximum number of layers that can be created for any gang-scheduled partition; any attempt to run an application that would cause this limit to be exceeded is rejected.

Layers are scheduled as a unit: all the applications and subpartitions in a layer are rolled in or out at the same time. The allocator examines the layers of each gang-scheduled partition at the end of each rollin quantum and whenever an application starts or terminates in the partition or one of its subpartitions. The allocator compares the priorities of the applications and subpartitions in each layer to determine which layer to run. Under some circumstances it may also move applications or subpartitions to different layers in order to reduce the number of layers and increase the number of running applications. However, once a set of nodes has been allocated to an application, the allocator never moves the application to a different set of nodes. Layering can sometimes result in applications being rolled in and out in counterintuitive ways.

For example, suppose the compute partition has four nodes and is gang-scheduled with a rollin quantum of 0. Now suppose that three users (*user1*, *user2*, and *user3*) start up applications as follows:

```
user1> app1 -sz 2
user2> app2 -sz 2
user3> app3 -sz 1
```

If the commands are issued in the order shown here, the applications will be scheduled as follows (also see Figure 8-1):

1. When *app1* begins, the allocator creates a scheduling layer (layer #1) and places *app1* on the first available nodes in that layer: nodes 0 and 1. Because there are no other layers, the allocator activates layer #1 and *app1* begins running immediately.

2. When *app2* begins, the allocator inspects layer #1 and places the application on the first two available nodes in that layer: nodes 2 and 3. Because *app2* is in the currently-active scheduling layer, it begins running immediately.

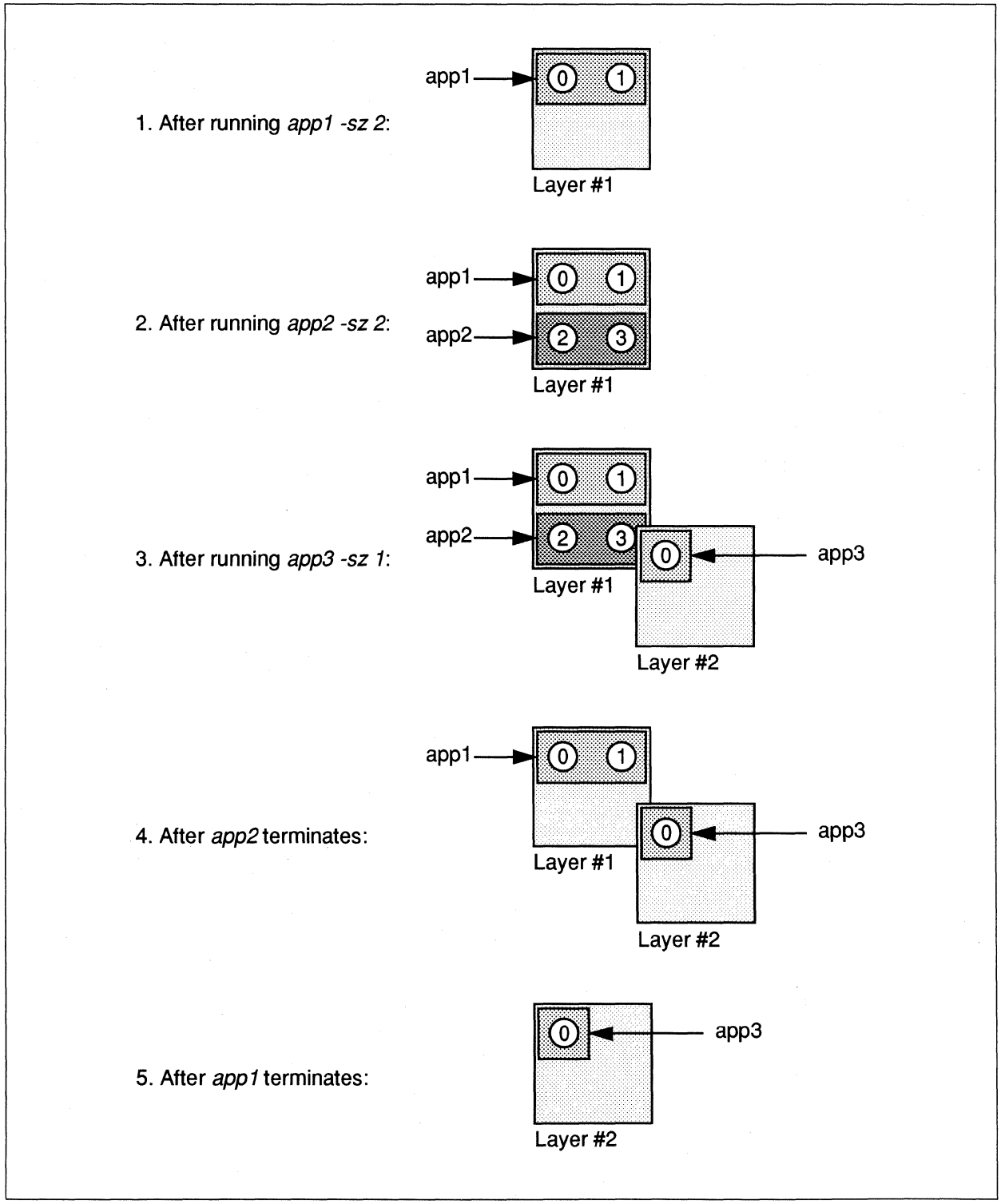


Figure 8-1. Example of Layers

3. When *app3* begins, the allocator inspects layer #1 and discovers that there is no room for the new application there. It creates a new scheduling layer (layer #2) and places *app3* in the first available node of that layer: node 0. Because *app3* is not in the currently-active scheduling layer, it does not begin running.

   At this point *app1* and *app2* are running (rolled in) in layer #1, while *app3* is waiting (rolled out) in layer #2.

4. Now suppose that *app2* finishes, but *app1* is still running. You might expect that *app3* would begin running at this point, because *app3* is a one-node application and there are two nodes free. However, *app3* does not begin running, because the allocator placed it on node 0 when it started and node 0 is still busy. Nodes are allocated to applications when the application is invoked, not when it actually begins running.

5. When *app1* finishes, the allocator notices that layer #1 is empty and removes it. It then activates layer #2 and *app3* begins running.

   When *app3* terminates, the allocator removes layer #2.

This is a very simple example; the allocator's behavior with larger applications and more complex layering may be more difficult to understand and predict.

Guidelines for Configuring Partitions

The root, service, compute, and I/O partitions and the allocator configuration file are created and maintained by the system administrator. You should configure these system features to provide the desired level of system services and enforce your site's system usage policies. In particular, the allocator configuration file and the permissions of the compute partition determine how users are allowed to run applications. This section provides some guidelines and recommendations to help you determine the best partition configuration for your system.

Recommended Partition Configurations

Running overlapping applications in gang-scheduled partitions is a known source of system instability. Therefore, Intel Scalable Systems Division recommends that you use one of the following two partition configurations for maximum system stability:

- The *space-sharing configuration*, which does not allow gang-scheduled partitions.
- The *NQS configuration*, which allows one gang-scheduled partition (the compute partition).

If you use NQS in both prime time and non-prime time and use different partition configurations in the two time periods, you should use the NQS configuration; otherwise, use the space-sharing configuration. The rest of this section tells you how to configure your system for either configuration. (Most partition attributes are the same in both configurations; differences are pointed out where necessary.)

You can also use a *gang-scheduled configuration*, which allows gang scheduling anywhere in the system. However, this configuration is not as stable as the space-sharing configuration or the NQS configuration, so it is not recommended and is not discussed in this chapter.

The Space-Sharing Configuration

The *space-sharing configuration* uses the allocator's *SPACE_SHARE* parameter to prevent overlapping applications and partitions. Once you have set the *SPACE_SHARE* parameter to 1, you can set the permissions of the compute partition to any desired value for your site. For example, you could set its permissions to 755 to allow users to run applications directly in the compute partition but not create subpartitions in the compute partition. See "The Compute Partition" on page 8-21 for more information.

The NQS Configuration

The *NQS configuration* uses the Network Queueing System (NQS) to control access to part of the compute partition; the remainder is allocated to a partition called the *interactive partition*. In this configuration, a user who wants to run an application can either submit the application to NQS or run the application in the interactive partition.

For example, assuming that the NQS queue *p32* is a 32-node queue, the user could use the following command to run the job script *myjob* on 32 nodes using NQS:

```
% qsub -q p32 myjob
```

If all the parameters are acceptable (that is, the queue the job was submitted to exists), NQS returns a request ID. The user can use the **qstat** command to check the status of the job, and **qdel** to remove the job from the queue. See the *Paragon™ System Network Queueing System Manual* for more information on using NQS.

Alternatively, assuming the interactive partition is called *OPEN*, the user could use the following command to run the application *myapp* on 32 nodes of the interactive partition:

```
% myapp -sz 32 -pn OPEN
```

The advantages of the NQS configuration are that NQS queues applications, makes sure the applications are run, and automatically resubmits jobs that don't run because the request was denied due to an unavailable partition. The disadvantage of this configuration is that unless the complete compute partition is under NQS control, the size of the applications you can run is diminished.

See "Configuring NQS" on page 8-31 for more information on the NQS configuration.

The Allocator Configuration File

The allocator configuration file, */etc/nx/allocator.config*, determines what limitations the allocator places on gang-scheduled partitions. See "The Allocator Configuration File" on page 8-10 for information on this file. Intel Scalable Systems Division recommends that this file contain the following lines:

- For the space-sharing configuration:

```
SPACE_SHARE=1
```

- For the NQS configuration:

```
NUM_GANG_PARTS=1
DEGREE_OF_OVERLAP=2
MIN_RQ_ALLOWED=1h
```

- For either configuration, if you use MACS you should also include the following line:

```
USE_MACS=1
```

  If you do not use MACS, the *allocator.config* file should contain *only* the lines shown above.

In the NQS configuration, the value of *MIN_RQ_ALLOWED* may have to be adjusted for your system; see "Determining the Minimum Rollin Quantum" on page 8-14 for information on determining the correct value of this parameter.

The Root Partition

The root partition must contain every usable node in the system. It determines the node numbering used by the OS (nodes are identified by their position within the root partition, which is called the *OS node number* or *root partition node number*). If there are any nodes outside the root partition, they will not be recognized by operating system and will not have OS node numbers.

The root partition is always rectangular in shape. If there are any empty slots or failed nodes within this rectangle they are included in the root partition, and given a number, but are marked as unusable by the allocator.

The root partition should use space sharing. It should be owned by user *root* and group *daemon*, and its permissions should be set to 754 (`rwxr-xr--`). This allows only *root* to modify the partition or create subpartitions, and allows only *root* or *daemon* processes to run directly in the partition. Ordinary users can only read the partition's attributes.

The only subpartitions of the root partition should be the service partition, compute partition, and optional I/O partition. These partitions should not overlap each other, and should together include all the usable nodes in the root partition.

The Service Partition

The service partition should consist of sufficient nodes to support your system's interactive load. The exact number of nodes depends on the amount of memory on each service node, the number of users on the system at once, the number and types of programs those users run in the service partition (such as shells, editors, and compilers), and any other services provided by service nodes (such as I/O). You can use **spv** or the **vm_stat** command to determine how much memory is in use on each service node; this is a good indication of whether or not the node is overloaded.

The service partition must use standard scheduling. It should be owned by user *root* and group *daemon*, and its permissions should be set to 754 (`rwxr-xr--`). Note that these permissions only prevent users from running *parallel* applications in the service partition; standard non-parallel programs such as shells and editors are not controlled by the allocator, so they are not affected by the partition's permissions.

Because the service partition uses standard scheduling, it cannot have any subpartitions.

CAUTION

The service partition *must* include the boot node.

Any other I/O nodes can be placed in the service partition, in the compute partition, or in a separate I/O partition, as discussed later in this section. If you place your I/O nodes in the service partition, they must perform I/O as well as running user processes, so you should consider increasing the number of nodes per user in the service partition.

NOTE

Changes to the size or shape of the service partition do not take effect until the system is rebooted.

This occurs because the operating system determines which nodes are in the service partition *only* at boot time, as discussed under "Service Partition Initialization at Boot Time" on page 8-7. If you change the size or shape of the service partition (by removing it with **rmpart** and then re-creating it with **mkpart**), the **showpart** or **lspart** command will reflect the change immediately. However, the operating system and load leveler will continue to use the nodes of the old service partition until the next time the system boots.

The I/O Partition

The service partition must include the boot node, but you can place any I/O nodes other than the boot node in the service partition, in the compute partition, or in a separate I/O partition. The advantage of a separate I/O partition is that you can restrict its permissions so that no user processes can run on those nodes. Doing this can increase I/O efficiency and make I/O performance more consistent by eliminating all load on the I/O nodes other than system I/O processes. The disadvantage of a separate I/O partition is that those nodes cannot be used for other work—when there is no I/O, they sit completely idle and do not contribute to the performance of the system.

If you configure your system with an I/O partition, the I/O partition should use standard scheduling. It should be owned by user *root* and group *daemon*, and its permissions should be set to 754 (`rwxr-xr--`). Note that user I/O processes are not controlled by the allocator, so they are not affected by the partition's permissions.

The I/O partition should not have any subpartitions.

The Compute Partition

All nodes that are not part of the service or I/O partition should be placed in the compute partition. The compute partition should be owned by user *root* and group *daemon*.

Any I/O nodes other than the boot node can be placed in the service partition, in the compute partition, or in a separate I/O partition, as discussed earlier in this section. If you place your I/O nodes in the compute partition, they must perform I/O as well as running user processes, which can impact both I/O performance and compute performance. However, I/O performance for user processes running on the I/O nodes themselves may actually be improved.

Intel Scalable Systems Division recommends that the compute partition's scheduling type and permissions be set as follows:

- For the space-sharing configuration, the compute partition should use space sharing and its permissions should be set to an appropriate value depending on the policies of your site. Typical values for these permissions are:

755	Allows users to run applications in the compute partition, but not create subpartitions of the compute partition.
766	Allows users to create subpartitions of the compute partition, but not run applications in the compute partition. To run an application, the user must make a subpartition, change its permissions to allow execution, then run the application in the subpartition.
777	Allows users to run applications or create subpartitions in the compute partition.

- For the NQS configuration, the compute partition should use gang scheduling with a rollin quantum of 0, and its permissions should be set to 744. These permissions do not allow users to run applications or create subpartitions in the compute partition; users must run their applications in subpartitions of the compute partition that have been created for them by the system administrator or by a process running as *root* (such as NQS).

Creating a New Partition Configuration

NOTE

For information about using the new **-l**, **-p**, and **-nt** switches in the **mkpart, lspart**, and **showpart** commands to configure node boards (General Processor (GP) boards, or Multi-Processor (MP) boards) refer to "Configuring Node Boards" on page 8-30.

This section tells you how to create the root, service, I/O, and compute partitions and the allocator configuration file from scratch. Use the procedure in this section if:

- Your system is being set up for the first time.
- Your system has had a configuration change that invalidated the root partition, such as adding or removing a cabinet (see "Root Partition Initialization at Boot Time" on page 8-8).
- The partition configuration files under */etc/nx* have been deleted or corrupted (see "Partition Configuration Files" on page 8-5).
- You want to completely remove your current partition configuration and start over with a new one. For example, you might do this if you have added some I/O nodes and want to change the service and I/O partitions accordingly.

The examples in this section assume a 64-node system configured as an 8-by-8-node rectangle with the boot node at node 7 and additional I/O nodes at nodes 31 and 39.

CAUTION

This procedure will delete all partitions and terminate all running applications. Do not use this procedure unless you are sure you want to remove your current partition configuration and create a new one from scratch.

1. Log into the Paragon system as *root*.

2. Create the file */etc/nologin* to prevent new users from logging in:

```
# cat > /etc/nologin
Logins disabled to reconfigure partitions
<Ctrl-D>
```

3. Use the **wall** command to ask all current users to log off:

```
# wall
Reconfiguring partitions, please log off now
<Ctrl-D>

Broadcast Message from root@super (console) at 18:39 ...

Reconfiguring partitions, please log off now
```

4. Wait a minute, then use the **who** command to verify that all users have logged off:

```
# who
root         console       Mar 18 17:44
```

If any users remain, use the **talk** command to ask them to log off. Do not proceed until you are the only user on the system.

5. If necessary, shut down NQS:

```
# qmgr
Mgr: shutdown
Mgr: exit
# /sbin/init.d/nqs stop
```

6. If necessary, shut down MACS:

```
# si -on -c "Reconfiguring partitions"
# /sbin/init.d/macs stop
```

7. Remove all partitions other than the root partition (if any).

CAUTION

This step will remove all partitions and terminate all running applications.

For example:

```
# lspart .
 USER     GROUP    ACCESS  SIZE    FREE  RQ   EPL  PARTITION
 root     daemon      755     1       1   -     -   service
 root     daemon      755     2       2   -     -   io
 root     daemon      777    61      61   1h    5   compute
# rmpart -r -f .service
# rmpart -r -f .io
# rmpart -r -f .compute
```

8. Remove the file */etc/nx/.partinfo* to invalidate the root partition:

```
# rm /etc/nx/.partinfo
```

9. Use the **initpart** command to make a new root partition based on the current bootmagic values and a new *allocator.config* file with default values:

```
# /sbin/initpart
```

10. Edit the file */etc/nx/allocator.config* as follows (see "The Allocator Configuration File" on page 8-10 for information on this file):

    - For the space-sharing configuration, *allocator.config* should contain the following line:

      ```
      SPACE_SHARE=1
      ```

    - For the NQS configuration, *allocator.config* should contain the following lines:

      ```
      NUM_GANG_PARTS=1
      DEGREE_OF_OVERLAP=2
      MIN_RQ_ALLOWED=1h
      ```

    - For either configuration, if you use MACS you should also include the following line:

      ```
      USE_MACS=1
      ```

      If you do not use MACS, the *allocator.config* file should contain *only* the lines shown above.

    In the NQS configuration, the value of *MIN_RQ_ALLOWED* may have to be adjusted for your system; see "Determining the Minimum Rollin Quantum" on page 8-14 for information on determining the correct value of this parameter.

11. Stop the System Monitor Daemon (SMD):

```
# /sbin/init.d/smd stop
```

12. Stop and re-start the allocator, to be sure the allocator knows about the new root partition and *allocator.config* file:

```
# /sbin/init.d/allocator stop
# /sbin/init.d/allocator start
Partition services provided.
```

13. Use the **showpart** command to verify that the new root partition includes all functioning nodes in the system. For example:

```
# showpart .
 USER      GROUP     ACCESS  SIZE          FREE  RQ    EPL
 root      daemon       754    64            64 SPS      5
    +-----------------+
  0| * * * * * * * * |
  8| * * * * * * * * |
 16| * * * * * * * * |
 24| * * * * * * * * |
 32| * * * * * * * * |
 40| * * * * * * * * |
 48| * * * * * * * * |
 56| * * * * * * * * |
   +-----------------+
```

If the root partition does not include all the nodes you expect it to, check the *SYSCONFIG.TXT*, *MAGIC.MASTER*, and *BADNODES.TXT* files on the diagnostic station. Missing nodes may indicate a hardware problem; if this occurs, use the diagnostics to track it down.

14. If necessary, use the **chpart** command to change the root partition's attributes to match your site's policies. See "The Root Partition" on page 8-19 for guidelines on configuring the root partition.

NOTE

The attributes of the root partition will be the default values for the attributes of the service, I/O, and compute partitions.

15. Use the **mkpart** command to create the service partition, which must include the boot node. The service partition must have the name *.service*. Use the **-ss** switch to specify standard scheduling, and the **-nd** switch to specify the nodes you want. See "The Service Partition" on page 8-20 for guidelines on configuring the service partition. For example, to allocate nodes 7, 15, and 23 to the service partition:

```
# mkpart -nd 7,15,23 -ss .service
```

16. If you want an I/O partition, use the **mkpart** command to create it. The I/O partition should have the name *.io*. Use the **-ss** switch to specify standard scheduling, and the **-nd** switch to specify the nodes you want. See "The I/O Partition" on page 8-21 for guidelines on configuring the I/O partition. For example, to name the I/O partition *.io* and allocate nodes 31 and 39 to it:

```
# mkpart -nd 31,39 -ss .io
```

17. Use the **mkpart** command to create the *.compute* partition from all remaining nodes. Use the **-sz** switch to specify the size of the compute partition, which should be the size of the root partition minus the sizes of the service and I/O partitions; the allocator will automatically select the nodes that have not yet been allocated to a partition. See "The Compute Partition" on page 8-21 for guidelines on configuring the compute partition.

    - For the space-sharing configuration, the compute partition should use space sharing (the default) and its permissions should be set to 755. For example:

```
# mkpart -sz 60 -mod 755 .compute
```

    - For the NQS configuration, the compute partition should use gang scheduling, with a rollin quantum of 0, and its permissions should be set to 744. For example:

```
# mkpart -sz 60 -rq 0 -mod 744 .compute
```

18. After creating the compute partition, use the **showpart** command to display the nodes allocated to each partition. Verify that, together, the partitions include every node in the system (that is, each node is allocated to exactly one partition). For example:

```
# showpart .service
 USER      GROUP     ACCESS  SIZE          FREE  RQ    EPL
 root      daemon       754     3             3   -      -
    +-----------------+
  0| . . . . . . . * |
  8| . . . . . . . * |
 16| . . . . . . . * |
 24| . . . . . . . . |
 32| . . . . . . . . |
 40| . . . . . . . . |
 48| . . . . . . . . |
 56| . . . . . . . . |
    +-----------------+
# showpart .io
 USER      GROUP     ACCESS  SIZE          FREE  RQ    EPL
 root      daemon       754     2             2   -      -
    +-----------------+
  0| . . . . . . . . |
  8| . . . . . . . . |
 16| . . . . . . . . |
 24| . . . . . . . * |
 32| . . . . . . . * |
 40| . . . . . . . . |
 48| . . . . . . . . |
 56| . . . . . . . . |
    +-----------------+
# showpart .compute
 USER      GROUP     ACCESS  SIZE          FREE  RQ    EPL
 root      daemon       766    60            60 SPS     5
    +-----------------+
  0| * * * * * * * . |
  8| * * * * * * * . |
 16| * * * * * * * . |
 24| * * * * * * * . |
 32| * * * * * * * . |
 40| * * * * * * * * |
 48| * * * * * * * * |
 56| * * * * * * * * |
    +-----------------+
```

If the partitions do overlap, use the **rmpart** command to remove the overlapping partitions and use **mkpart** to re-create them. When you create the partitions again, you may need to use the **-nd** switch to specify the node numbers of the nodes to include in the compute partition. For example:

```
# rmpart .compute
# mkpart -nd 0..6,8..14,16..22,24..30,32..38,40..63 \
-mod 766 .compute
```

19. For the NQS configuration, create interactive partitions and configure NQS, as described under "Configuring NQS" on page 8-31. Skip this step for the space-sharing configuration.

20. If desired, use the **mkpart** command to create any additional subpartitions of *.compute* you want. For example, to create a partition called *.compute.small* with 5 nodes and a protection mode of 755:

```
# mkpart -sz 5 -mod 755 small
```

21. Remove the */etc/nologin* file to re-enable logins:

```
# rm /etc/nologin
```

22. Shut the system down:

```
# cd /
# shutdown now
# halt
```

23. On the diagnostic station, reboot the Paragon system:

```
DS# cd /usr/paragon/boot
DS# ./reset
```

NOTE

You *must* reboot the system each time you change the nodes allocated to the service partition.

Your partition configuration is now complete. Do not go on to the next section.

Balancing the Load in the Service Partition

The **loadlevel** command balances the system load across all service nodes of a Paragon system. The **loadlevel start** command executes automatically during multi-user start up.

Only the system administrator can run the **loadlevel** command. You may want to remove the load leveler service when:

- You want to eliminate static load leveling.
- You want to run the load level service only on a subset of the nodes.
- You want to set up different parameters for load leveling.

You can enter the command with one of two options:

```
# /sbin/init.d/loadlevel [start | stop]
```

The following command finds the processes for the load-leveler daemons and kills them.

```
# /sbin/init.d/loadlevel stop
```

You can also disable (or enable) static load-leveling on service nodes using the **f** switch on the `ENABLE_FORK_REMOTE` bootmagic string.

```
ENABLE_FORK_REMOTE=f
```

When you boot the system, this variable would then disable the load leveling service. Use the **t** switch in the variable to enable static load leveling (default).

To perform load leveling for a subset of service nodes, copy the */etc/load_level/parameters* file into an alternate parameter file (e.g., *altparamfile*) and edit the `nodes_to_use` parameter in the alternate parameter file to specify a subset of nodes. For example:

```
nodes_to_use=1,2,3
```

Refer to the manpage for the **parameters** command for more information about node selection and syntax. Once you have made changes and closed the alternate parameter file, enter the following command:

```
# /sbin/init.d/loadlevel -s -p altparamfile
```

The **-p** switch loads the named parameter file (*altparamfile*, in this example), and the **-s** option specifies static load leveling for each node listed in the `nodes_to_use` parameter.

Configuring Node Boards

Node boards are either General Processor (GP) boards, or Multi-Processor (MP) boards. You can use partition commands to configure GP systems, MP systems, or both. Refer to the *Paragon™ System User's Guide* for detailed descriptions of each type. To display general information about your system, use the **lspart** and **showpart** commands.

lspart lists all of the subpartitions, while **showpart** lists the characteristics of specific partitions. The **-l**, **-p**, and **-nt** switches to these commands show information that is specific to the node types:

NOTE

For more detailed information about these commands, refer to the *Paragon™ System User's Guide* and the Manual pages.

The following example shows an **lspart -l** command and resulting display:

```
% lspart -l .compute

USER      GROUP     ACCESS  SIZE          FREE  RQ    EPL  PARTITION  ATTR
terry     users        777     1             1 SPS      5  terry1     1proc,16mb,GP
```

The example above shows a *.compute* partition with one partition named *terry1*. Notice that it uses a 16M-byte General Purpose (GP) node board.

The following example shows a **showpart -p** command and resulting display in the *.service* partition:

```
% showpart -l .service
USER      GROUP     ACCESS  SIZE          FREE  RQ    EPL  ATTR
root      daemon       754     2             2   -      -  1proc,net,enet,MAXTOR
                                                          MXT-1240S I1.2,scsi,disk,io
0 1proc,64mb,MP,net,enet,MAXTOR MXT-1240S I1.2,scsi,disk,io
1 1proc,32mb,GP,net,enet,MAXTOR MXT-1240S I1.2,scsi,disk,io,bootnode

  +---------+
0 | . . . . |
4 | . . . . |
8 | . . . . |
2 | * . . * |
  +---------+
```

The following example isolates and displays only the positions of the 32MB nodes in the partition called *mypart*:

```
% showpart -nt 32mb mypart

smith     eng          777      9           5   15m     5
   +---------+
0  | . . . . |
4  | . * * * |
8  | . o * * |
12 | . o o o |
   +---------+
```

Nodes in the partition having the attributes specified in the *nodetype* (-nt 32mb) string are shown with an asterisk (*); other nodes within the partition are shown with a lowercase letter O (o).

You can use the **mkpart** command to create a partition. It lets you specify most of the characteristics of the partitions. You can use the **mkpart -nt** command switch to create a partition that consists only of specific nodes. For Example, to create a 25-node partition of MP nodes called mypart, enter the following:

```
% mkpart -sz 25 -nt mp -rlx mypart
```

The **-rlx** switch makes sure the partition is created even though there may not be 25 MP nodes available. If there are fewer than 25 MP nodes available, the partition is created with the available MP nodes.

NOTE

In the current release, the only supported configuration is for one processor on each node to be a message co-processor. This means that all nodes are mcp nodes, GP nodes are always 1proc, and MP nodes are always 2proc.

For more detailed information about the partition commands, refer to the *Paragon™ System User's Guide* and the Manual pages.

Configuring NQS

For the NQS configuration, you need to create two subpartitions of the compute partition for interactive use, and configure NQS to use the rest of the nodes. This section shows an example NQS configuration; see the *Paragon™ System Network Queueing System Manual* for more information on NQS.

Questions About Your Configuration

Before you can configure NQS, you will need to answer some questions about the way you will use NQS at your site:

- System configuration questions:

  How much of the system will be reserved for interactive jobs? How much will be reserved for batch jobs? Will you want to change this mix from day (prime time) to night (non-prime time)? If so, how and at what times?

- NQS configuration questions:

  How many queues will you have? How many nodes will each queue have? What type of nodes will be in each queue (for example, not all nodes have the same amount of memory)? Where in the machine will the queues be located? When is each queue available (day, night, both)? What time limits will be imposed (how long may a job run)? How many concurrent jobs will be allowed per queue?

- Miscellaneous questions:

  May jobs running toward the end of non-prime time keep running into prime time? Is your site using MACS? How long should a job wait to run? Are large (grand challenge) jobs or small jobs more prevalent? Which should have precedence? Who can access which queues?

The answers to these questions will be determined by the policies of your site. The answers to these questions form the *specification* for your NQS configuration.

Example NQS Specification

Here is an example of a specification for an NQS configuration:

- The system is a 2-cabinet (128-node) system with both 16MB and 32MB nodes, configured as shown in Figure 8-2. Eight nodes are used as service nodes.

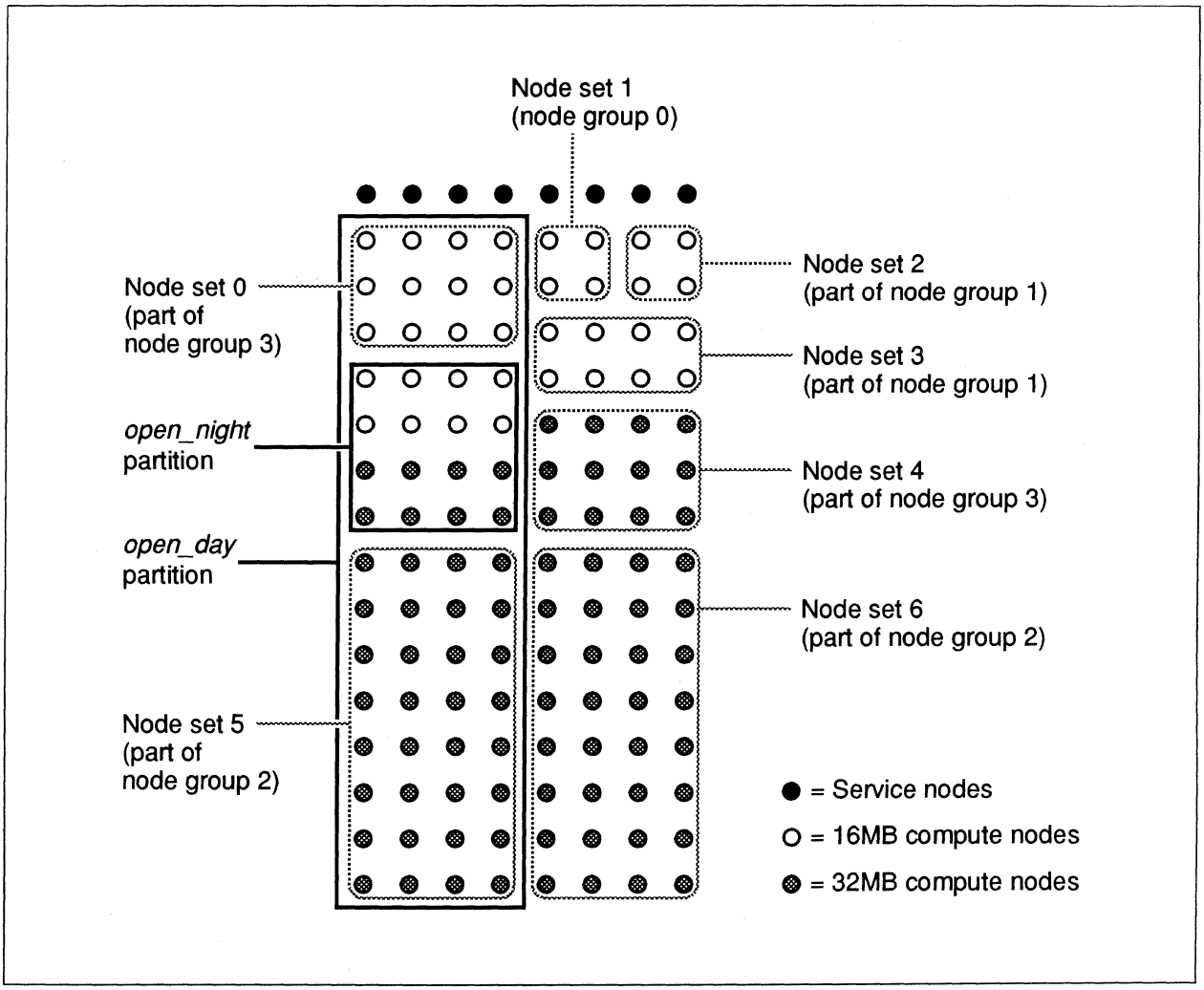


Figure 8-2. Example NQS Configuration

- During the day, half of the compute partition (60 nodes) shall be available for interactive use and half (60 nodes) for batch jobs.
- At night, 16 nodes shall be available for interactive use and the remainder (104 nodes) is used for batch jobs.

- NQS queues shall be defined so that, whenever possible, they do not share differing memory sizes within a single queue.
- The nodes available for batch jobs shall be divided into four queues as follows:

Queue 1	12 16MB nodes during both prime and non-prime time.
Queue 2	32 32MB nodes during prime time, 64 32MB nodes during non-prime time.
Queue 3	12 32MB nodes during prime time, 12 16MB nodes + 12 32MB nodes during non-prime time.
Queue 4	4 16MB nodes during both prime and non-prime time.

- Prime time is defined as 7:00 AM through 6:30 PM, 7 days per week.

This example is based on one set of answers to the questions in the previous section; your specification will probably be different.

Example NQS Configuration

In order to implement the specification in the previous section, the example configuration uses two interactive partitions (called *open_day* and *open_night*) and seven NQS node sets, as shown in Figure 8-2. Note the following characteristics of this configuration:

- The *open_night* partition is a strict subset of the *open_day* partition.
- The node sets do not overlap each other.
- All the nodes in each node set have the same amount of memory.
- No node set overlaps the *open_night* partition.
- No node set crosses the edge of the *open_day* partition.
- The node sets are collected into node groups in such a way that each node group can be assigned to a queue and different node sets can be used during prime time and non-prime time.

Implementing the Example NQS Configuration

The commands to configure the system as described in the previous section are as follows. This procedure should be performed as step 19 on page 8-28.

NOTE

This procedure assumes that NQS has already been installed.

See the *Paragon™ System Network Queueing System Manual* for information on installing NQS.

1. Make two subpartitions of the compute partition for interactive use, one for use during prime time and the other for use during non-prime time. For example:

```
# mkpart -nd 15x8:0 open_day
# mkpart -nd 4x4:24 open_night
```

2. Edit the file */usr/spool/nqs/conf/sched_param* to describe your desired NQS configuration, as follows:

   A. Set the **time_zone** parameter to your site's time zone, and the **prime_start** and **prime_end** parameters to the starting and ending times of prime time for each day of the week. For example:

```
time_zone : PST8PDT
prime_start : 07.00 07.00 07.00 07.00 07.00 07.00 07.00
prime_end : 18.50 18.50 18.50 18.50 18.50 18.50 18.50
```

   B. Set the **cur_open_name** parameter to **OPEN**, set the **open_p_name** parameter to the name of the interactive partition for prime time, and the **open_np_name** parameter to the name of the interactive partition for non-prime time. For example:

```
cur_open_name : OPEN
open_np_name : open_night
open_p_name : open_day
```

   These parameters tell NQS to use the name *OPEN* for the interactive partition, and to alternate this name between the partitions currently named *open_night* and *open_day*. At the beginning of prime time, NQS renames the partition *OPEN* to *open_night* and renames *open_day* to *OPEN*. Then, at the beginning of non-prime time, NQS renames *OPEN* to *open_day* and renames *open_night* to *OPEN*. This means that users can run applications interactively in a partition called *OPEN* at any time of the day or night; this partition will be a large one or a small one according to the time of day.

C. Set the **node_set***n* parameters to divide the nodes of the compute partition into node sets. For example (see Figure 8-2):

```
node_set0 : .compute 3x4:0
node_set1 : .compute 2x2:4
node_set2 : .compute 2x2:6
node_set3 : .compute 2x4:20
node_set4 : .compute 3x4:36
node_set5 : .compute 8x4:56
node_set6 : .compute 8x4:60
```

D. Set the **node_group***n* parameters to collect the node sets into node groups. For example (see Figure 8-2):

```
node_group0 : 1
node_group1 : 2 3
node_group2 : 5 6
node_group3 : 0 4
```

E. Set the **prime_list***n* and **nprime_list***n* parameters to specify which node sets in each node group are used during prime time and which are used during non-prime time:

```
prime_list0 : 1
prime_list1 : 2 3
prime_list2 : 6
prime_list3 : 4
nprime_list0 : 1
nprime_list1 : 2 3
nprime_list2 : 5 6
nprime_list3 : 0 4
```

F. Set the remaining NQS parameters as desired for your site. For example:

```
timeshare : 0
block_pri : -1
timesh_pri : 10
tsched_pri : 20
chk_runlimit : 0
age_factor : 2.0
grace_time : 10
rollin_quan : 600
do_wall : 0
np_overrun : 0
macs_flag : 0
```

3. Once the *sched_param* file is edited and saved, you must restart NQS to have the changes take effect. To do this, use the following command:

```
# /sbin/init.d/nqs start
```

4. Once NQS has restarted, use the **qmgr** command to create NQS queues using the node groups you have established. For example:

```
# qmgr
Mgr> create batch_queue queue_1 priority=3
Mgr> enable queue queue_1
Mgr> start queue queue_1
Mgr> set node_group = 1 queue1
Mgr> set per_request ncpus = 12 queue1

Mgr> create batch_queue queue_2 priority=3
Mgr> enable queue queue_2
Mgr> start queue queue_2
Mgr> set node_group = 2 queue_2
Mgr> set per_request ncpus = 64 queue_2

Mgr> create batch_queue queue_3 priority=3
Mgr> enable queue queue_3
Mgr> start queue queue_3
Mgr> set node_group = 3 queue_3
Mgr> set per_request ncpus = 24 queue_3

Mgr> create batch_queue queue_4 priority=3
Mgr> enable queue queue_4
Mgr> start queue queue_4
Mgr> set node_group = 0 queue_4
Mgr> set per_request ncpus = 4 queue_4

Mgr> quit
#
```

These commands create four queues, called *queue_1* through *queue_4*, configured as described in the previous section. Users can use the **qsub** command to submit jobs to these queues at any time of the day or night, but some queues will be larger or smaller depending on the time of day.

5. NQS is now configured. Return to step 19 on page 8-28.

See the *Paragon™ System Network Queueing System Manual* and the **qmgr** manual page for more information on NQS.

Managing UFS and NFS File Systems 9

The Paragon™ operating system supports three file system types:

UFS	UNIX File System, the standard file system type.
NFS	Network File System, a file system type that represents a file system on another computer on the network.
PFS	Parallel File System, a file system type that is optimized for access by parallel processes. This file system type is unique to Paragon.

This chapter discusses UFS and NFS file systems. Refer to "Managing PFS File Systems" on page 10-1 for a discussion of PFS file systems.

The File System Tree

In UNIX, the file system can be viewed as a tree that consists of several mounted file systems. General users view the file system tree as a single entity, since the mounting scheme is transparent once it is established. However, as system administrator, you must understand the underlying structure of the tree.

Managing the File System Tree

The UNIX file system tree consists of a top *root* directory and a variable number of directories subordinate to that directory. A single file system is a complete directory structure, including its own *root* directory with subdirectories and files beneath it. File systems are attached to the file system tree by the **mount** command and are detached from the file system tree by the **umount** command. When you mount a file system, you specify a location (the mount point) where the file system is to reside within the file system tree. The kernel uses this information to locate and access the directories and files that comprise that particular file system. The root directory of a mounted file system is identical to its mount point.

On systems that support multiple file system types, the *root* directory of the file system tree is the root directory of the file system type that was mounted (by the system) at boot time. Only one root directory can exist on the system at any given time since the system uses the root directory as its source for system initialization and self-maintenance. Other subsequently mounted file system types (and their corresponding file system directories and subordinate files) are mounted subordinate to the initial root directory.

The Root File System

When you install the Paragon operating system, the software defines a particular *root* file system for the file system tree. The bootstrap software defines which root file system is to be mounted, and provides internal instructions for system initialization.

Typically, the *root* file system contains the *root* (*/*) directory and a small set of files and subdirectories that the system relies on during its bootstrap and initialization activities.

Throughout your work as system administrator, you use the system files and directories that reside within the root file system. The following list describes some of the major files and directories that comprise the root file system, and that are of particular value to you in your work:

- The */sbin* directory contains the binaries required to successfully boot and initialize the system in single user mode.
- The */dev* directory contains device files and subordinate directories for input and output operations to physical and pseudo devices.
- The */etc* directory contains most of the standard system administration files, and some commands used for startup, shutdown, and internationalization. The executables are linked to commands in the */usr/sbin* or */usr/share/lib* directories.
- The */tmp* directory contains temporary files.

Do not change the contents of the *root* file system directories; they should remain as they existed when installed. The system relies on specific paths defined within the source code. Changing the location of a binary can cause failure or an unpredictable response in command execution.

Since the commands that reside within the *root* file system directories are critical for successful system use, you should make certain that you include these directories in the *PATH* variable of your startup file.

The */sbin* Directory

The */sbin* directory contains a small set of executables required by the system to successfully boot and initialize the system in single-user mode. Certain files in this directory provide system recovery mechanisms as well.

The */bin* and */usr/bin* Directories

The */bin* directory is a linked directory that points to the */usr/bin* directory. The */usr/bin* directory contains additional UNIX executables and linked files.

The */dev* Directory

The */dev* directory contains device (or "special") files. Device files are of particular importance within the system environment since they serve as the link between the system and the device drivers. Each device file corresponds to a physical device (a disk, tape, printer, or terminal, for example) or a pseudo device (a network interface, a named pipe, or domain socket, for example). The driver knows how to handle all read and write operations that relate to the specific device, and will follow the required protocols for that device. For information about managing I/O devices in the */dev* directory, refer to "The /dev Directory" on page 9-3.

The /etc Directory

The */etc* directory contains several of the administrative files that you use in your work. For example, the files associated with disks, mounts, network services, terminals, accounting, and so forth reside within this directory. Some of the administrative files that define internationalization functionality are located within */etc/nls*.

The */etc* directory contains regular files, executables, and linked files. Most of the files in this directory are owned by *root*, although some are owned by a daemon process such as *bin*.

The /usr File System

Another important file system that you require for system administration tasks is the */usr* directory, which consists of several subordinate directories (and linked directories) containing such files as local UNIX commands, on-line documentation, libraries, spooling directories, administrative logging files, and so forth.

The system mounts the */usr* file system during initialization because, although it is not required for a successful boot to single user mode, its availability is critical for administrative tasks and during subsequent operations. For example, */usr* contains specific libraries, spooling directories, and administrative files that the system relies on after the boot operation.

You also rely on this file system in your work. For example, you check logging files from within the *adm* directory, set up additional print spooling directories and files subordinate to the *spool* directory, and read on-line documentation by way of operations involving files subordinate to the *man* directory. Although it is not mandatory, administrators frequently mount user file systems onto the */usr* directory.

This file system typically contains directories and links to directories. For example:

```
lrwxr-xr-x   1 root    system        10 Apr 20 16:33 adm -> ../var/adm
drwxr-xr-x   2 root    system       512 Apr 20 17:18 backups
drwxr-xr-x   3 root    system      5632 Apr 14 17:32 bin
drwxrwxr-x   4 root    system       512 Apr 14 14:19 ccs
drwxr-xr-x   3 root    system       512 Apr 14 02:09 dict
drwxr-xr-x  31 root    system      3584 Apr 14 17:37 include
drwxrwxr-x   4 root    system       512 Apr 14 02:16 lbin
drwxr-xr-x  10 root    system      1536 Apr 14 17:37 lib
drwxr-xr-x   2 root    system       512 Apr 20 17:18 local
drwxrwxr-x   4 root    system       512 Apr 14 02:21 mach
lrwxr-xr-x   1 root    system        11 Apr 20 16:33 mail -> ../var/mail
lrwxr-xr-x   1 root    system         9 Apr 20 16:33 man -> share/man
lrwxr-xr-x   1 root    system        11 Apr 20 16:33 news -> ../var/news
lrwxr-xr-x   1 root    system        15 Apr 20 16:33 preserve -> ../var/preserve
drwxrwxr-x   3 root    system      2048 Apr 01 1993  sbin
drwxrwxr-x   9 root    system       512 Apr 14 17:37 share
lrwxr-xr-x   1 root    system        12 Apr 20 16:33 spool -> ../var/spool
drwxrwxrwt   2 root    system       512 Apr 26 10:05 tmp
lrwxr-xr-x   1 root    system         3 Apr 20 16:33 ucb -> bin
```

The Structure of File Systems

Regardless of the device on which they reside, all file systems have the same basic structure. A file system has four major parts:

- Bootstrap block.
- Superblock.
- I-node blocks.
- Data blocks.

The first block of every file system (block 0) is reserved for a bootstrap, or initialization, program. The bootstrap block is not actually part of the file system structure. File system data begins on block 1, the superblock.

The Superblock

Block 1 of every file system is called the superblock, and contains the following information:

- The total size of the file system (in blocks).
- The number of blocks reserved for i-nodes.
- The name of the file system.
- The device identification.
- The date of the last superblock update.
- The head of the free-block list, a chain that contains all of the free blocks in the file system (the blocks available for allocation). When new blocks are allocated to a file, they are allocated from this list. When a file is deleted, its blocks are returned to this list.
- A list of free i-nodes, the partial listing of i-nodes available to be allocated for newly created files.

I-nodes

A group of blocks follows the superblock. Each of these blocks contains a number of i-nodes. Each i-node describes an individual file in the file system. There is one i-node for each possible file in the file system. Each i-node is associated with an i-node number, or i-number. For any file system, there is a maximum number of i-nodes, which dictates a maximum number of files that the file system can contain. This maximum number varies with the size of the file system. The first i-node (i-node 1) on every file system is unnamed and unused. When associated with files, the remaining i-nodes contain the following information:

- File type. Possible file types are regular file, directory file, device file, FIFO, socket, and symbolic link.
- File owner. The i-node contains the user and group IDs associated with the file.
- Protection information. This is a specification of read, write, and execute access for the owner, members of the group associated with the file, and others. It also includes other mode information specified by the **chmod** command.
- Link count. A directory entry (link) consists of a name and the number of the i-node that represents the file (its i-number). The link count specifies the number of directory entries that refer to the file. A file is deleted when the link count becomes zero. That is, its i-node is returned to the list of free i-nodes and its associated data blocks are returned to the free-block list.
- Size of the file (in bytes).

- Date the file was last accessed.
- Date the file was last modified.
- Date the i-node was last modified.
- Pointers to data blocks . These pointers indicate the location of the data blocks on the physical disk.

I-node 2 must correspond to the root directory for the file system. All other files in the file system are below the root directory in the file system. Above i-node 2, any i-node can be assigned to any file. Similarly, any data block can be assigned to any file. Neither i-nodes nor blocks have to be allocated in any particular order.

Managing File System Directories and Files

The operating system views files as bit streams, which allows you to define and handle on-disk data, named pipes, domain sockets, terminals, and so forth as files. This object type transparency offers a simple mechanism for defining and working with a wide spectrum of storage and communication facilities. The operating system handles the various levels of abstraction as it organizes and manages its internal activities.

While it is true that you deal only with the external interface, it is useful to understand the various file types recognized by the system.

The system supports several file types, namely:

- Regular files. A regular file contains data in the form of a program, a text file, or source code, for example.
- Directories. A directory is a type of regular file that contains a list of other, subordinate files that reside beneath it.
- Character and block device files. These files identify physical and pseudo devices on the system. See "The /dev Directory" on page 9-3 for details.
- UNIX domain socket files. Domain socket files provide the connection between network processes. The socket files are created with the **socket** system call. Refer to the *OSF/1 System and Network Administrator's Reference* for details.
- Named pipes. Named pipes are device files used by communication processes operating on the same host machine. Refer to "Mounting File Systems" on page 9-7 for details on creating this type of device file and to the *OSF/1 System and Network Administrator's Reference* for information about communication processes.

- Symbolic link files. A symbolic link file is one that contains the name of another file (or directory) to which it points. The target file is the file to which another file points. A symbolic link file and its target file may reside on the same file system or they may be on different file systems. A symbolic link file differs from a hard link in that hard links must exist on the same file system.

Creating and Deleting Regular Files

As mentioned previously, a regular file is simply a file that contains data. There are several commands that you use to create regular files. To create a regular file, use one of the available editors, **vi** or **ed**, for example. When the system first boots to single user mode, only the **ed** editor is available; subsequently, additional editors are available.

For more information about these commands, refer to the *OSF/1 Command Reference*.

Mounting File Systems

You attach a file system to the file system tree by *mounting* it. The **mount** command makes this connection.

Each file system is stored on a specific device, and must be connected to a specific directory (called the *mount point*) that currently exists on the system. The **mount** command connects a device (specified by its device file in */dev*) to a directory (specified by its absolute pathname). Because a device file can specify a device on a remote node (see "The /dev Directory" on page 9-3), you can mount a file system from one node onto a directory in a file system on a different node. This feature transparently melds all the nodes' file systems into a *single system image*. No special commands are required to mount or access files on other nodes.

- At boot time, the system automatically mounts file systems whose definitions exist in the */etc/fstab* file. This file contains entries that specify the device with which the file system is associated, the mount point, and additional information that defines the attributes of the file system. "File System Mounts" on page 9-8 describes the contents and notation of the /etc/fstab file. "Editing the /etc/fstab File" on page 9-10 provides instructions for editing this file.

  Currently, the operating system supports using only the UFS file system as the root file system. Consequently, the UFS file system is the root of the entire file system tree. You mount additional file system types subordinate to this root structure. As discussed earlier, the system supports only one root file system from which it accesses the executable kernel and other binaries and files that it needs to successfully boot and perform its initialization activities. The root file system mounted at boot time cannot be unmounted.

- At run time, you can mount additional file systems with the **mount** command.

File System Mounts

The */etc/fstab* file contains descriptive information about the known file systems. */etc/fstab* is read but not written by programs. The order of entries in */etc/fstab* is important because **fsck**, **mount**, and **unmount** sequentially iterate through */etc/fstab* during their work.

By convention, you create and maintain this file; consequently, you should familiarize yourself with its contents and meaning.

Contents of the /etc/fstab File

The */etc/fstab* file contains entries for all known file systems. The following */etc/fstab* file fragment provides a sample for you to examine.

```
/dev/io0/rz0a          /              ufs rw 0 1
/dev/io0/rz0e          /usr           ufs rw 0 2
/dev/io0/rz0f          /home          ufs rw 0 3
/dev/io0/rz0d          /pfs           pfs rw,stripegroup=four 0 3
mysun:/users/chris     /home/chris    nfs rw 0 0
```

As seen in the sample above, each file system is described on a separate line; fields on each line are separated by tabs or spaces. Table 9-1 describes the fields that make up each line.

note 1: If the option **userquota** or **groupquota** is specified, the file system is automatically processed by the **quotacheck** command, and disk quotas are enabled with the **quotaon** command. By default, file system quotas are maintained in files named *quota.user* and *quota.group* which are located at the root of the associated file system. These defaults may be overridden by putting an equal sign (=) and an alternative absolute pathname following the quota option. Thus, if the user quota file for */tmp* is stored in */var/quotas/tmp.user* this location can be specified as **userquota=/var/quotas/tmp.user**.

The type of the mount is extracted from the *fs_mntops* field and stored separately in the *fs_type* field. It is not deleted from the *fs_mntops* field. If *fs_type* is `rw` or `ro`, then the file system whose name is given in the *fs_file* field is normally mounted read-write or read-only on the specified special file. If *fs_type* is `sw`, then the special file is made available as swap space by the **swapon** command at the end of the system reboot procedure. The fields other than *fs_spec* and *fs_type* are unused. If *fs_type* is specified as `xx`, the entry is ignored. This is useful to show disk partitions which are currently unused.

Refer to the **fstab**(4) file in the *OSF/1 System and Network Administrator's Reference* for more details.

Table 9-1. Descriptions of */etc/fstab* Fields

Field Number	Field Name	Description
1	*fs_spec*	Describes the block special device or remote file system to be mounted. For file systems of type **ufs**, the special file name is the block special file name, and not the character special file name. If a program needs the character special file name, the program must create it by inserting the letter "r" after the last / (slash) in the special file name.
2	*fs_file*	Describes the mount point for the file system. For swap partitions, this field should be specified as **none**.
3	*fs_type*	Describes the type of the file system. The system currently supports four types of file systems: **ufs** (a local UNIX file system), **nfs** (a network file system), **pfs** (a Parallel File System), and **swap** (a disk partition to be used for swapping).
4	*fs_mntops*	Describes the mount options associated with the file system. It is formatted as a comma-separated list of options and contains at least the type of mount plus any additional options appropriate to the file system type (see note 1).
5	*fs_freq*	Used by the **dump** command to determine which file systems need to be backed up. If the fifth field is not present, a value of zero is returned and **dump** assumes that the file system does not need to be dumped.
6	*fs_passno*	Used by the **fsck** program to determine the order in which file system checks are done at reboot time. The value of this field should be determined as follows: 1 Root (/) file system. 2 File systems required for single-user operation. 3 Boot-node local file systems. 4 or more Remote-node file systems, NFS file systems, and PFS file systems. 0 File systems that don't need to be checked. File systems within a drive will be checked sequentially, but file systems on different drives will be checked at the same time to utilize parallelism available in the hardware. If this field is not present, a value of zero is returned and *fsck* assumes that the file system does not need to be checked.

Editing the /etc/fstab File

To edit the */etc/fstab* file, follow this sequence:

1. Log on as root and open the file with one of the available editors.
2. Edit the file using the mandatory format.
3. Write and close the file.

To enforce changes immediately, run the **mount -a** command after editing the */etc/fstab* file. The mount program examines */etc/fstab* and mounts additional file systems if necessary. However, previously mounted file systems continue to be mounted even though they may no longer be in the */etc/fstab* file. For example, if the file system entry for a previously mounted file system is deleted from */etc/fstab*, it remains mounted unless specifically unmounted. When the system is rebooted, the file systems that are automatically mounted with the **mount -a** command reflect what is listed in the */etc/fstab* file.

Run Time File System Mounts

There are instances where you do not need to manually edit the */etc/fstab* file to mount a file system. For example, you can make a temporary mount of a particular file system, and you can change information of an entry in the */etc/fstab* file without having to reboot the system. In these cases, you use the **mount** command with the available options. The mount takes effect immediately but it is not automatically reinstated if the system is subsequently shut down and rebooted.

The **mount** command takes several options and supports the UFS, NFS, and PFS file system types. Refer to the *OSF/1 System and Network Administrator's Reference* for a full description of the command, its options, and related information. The following sections briefly describe the command syntax and provide examples of command use for each file system type.

Listing Current Mounts

To find out which file systems are currently mounted, use the **mount** command without options. For example, enter mount with no flags:

```
mount
```

Root permission is not required to view the mounted file systems.

Mounting UFS File Systems

To mount specific UFS file systems at run time, log on as root and enter the **mount** command with this syntax:

```
mount [options] [device] [pathname]
```

- If you specify both *device* and *pathname*, the command mounts the specified device at the specified point in the file system. The *options*, if any, control the way the file system is mounted.
- If you specify a *device* without a *pathname* or a *pathname* without a *device*, the command searches for the specified device or pathname in the file */etc/fstab* and mounts it as specified in that file. The *options*, if any, control the way the file system is mounted.
- If you specify neither *device* nor *pathname*, the *options* select one or more file systems from the file */etc/fstab*. For example the command **mount -a** mounts all the file systems listed in */etc/fstab*; the command **mount -t ufs** mounts all file systems of type **ufs** listed in */etc/fstab*.
- If you don't use any arguments, the command lists all the currently-mounted file systems.

You can change the status of a previously-mounted file system with the **-u** switch. For example, to change the read-write access on */tools* (whose corresponding device is partition **g** of SCSI disk drive 0 on the boot node) to read-only, enter this command:

```
mount -u -r /dev/io0/rz0f /tools
```

Refer to the **mount** command in the *OSF/1 System and Network Administrator's Reference* for a description of the command and its options.

Mounting an NFS File System

Mounting an NFS file system is very similar to mounting a UFS file system. The **mount** command supports remote file system mounts. To mount a specific file system from a remote host at run time, log on as root and enter the **mount** command with this syntax:

```
mount -t nfs [options] rhost:directory directory
```

For example, to mount */usr/users/homer* (located on the remote host *acton*) on your local system at */users/homer* with read-write access, and to request that the mount be soft, enter this command:

```
mount -t nfs -w -o soft acton:/usr/users/homer /users/homer
```

To mount all NFS file systems listed in */etc/fstab*, enter the **mount** command with this syntax:

```
mount -t nfs [options]
```

Refer to the **mount** command in the *OSF/1 System and Network Administrator's Reference* for a description of the command and its options.

NOTE

See Chapter 10 for information on mounting PFS file systems.

Unmounting File Systems

To detach a file system from the file system tree with the **umount** command use this syntax:

```
umount [-a -f] [-t nfs | ufs | pfs] device | pathname
```

If the file system is in use when the command is invoked, the system returns an error message: `device busy`, and does not unmount the file system. The system takes a broad view of file system use. If any user process (including a **cd**) is in effect within the file system, the request to unmount that file system is denied. You cannot unmount the root file system with the **unmount** command.

Refer to the *OSF/1 System and Network Administrator's Reference* for a full description of the command, its options, and command use.

Creating File Systems

Since a file system is a complete directory structure that resides on a device, you create a file system with commands that require information about the format and values of the device on which the file system resides.

The **newfs** command formats a labelled disk partition into a file system. The command is a front end for the UFS file system **mkfs** commands and builds the new file system on a specific device, using information in the disk label as its default values. The command uses reasonable default values; nevertheless, you can override these values by using specific command options to redefine the standard sizes for disk geometry.

WARNING

Changing the default values makes it impossible for the **fsck** program to find the alternate superblocks if the standard superblock is lost.

To create a new file system, use the **newfs** command with this syntax:

```
newfs [-N] [options] special device
```

After you create a new file system, run the **fsck** program to confirm that you can access the disk and that the new file system is consistent.

For more information about the **newfs** command, refer to the *OSF/1 System and Network Administrator's Reference*.

Tuning File Systems

To enhance the efficiency of UFS file system reads, you can tune the file system with the **tunefs** program. The syntax for the command is:

tunefs *options special_device | file system*

The following **tunefs** options specify the dynamic parameters which affect the disk partition layout policies:

-a *maxcontig* This option specifies the number of contiguous blocks that can be written before enforcing a rotational delay. The default value is 1.

-d *rotdelay* This option specifies the rotational delay (in milliseconds) to allow between successive blocks in a file.

-e *maxbpg* This option specifies the maximum number of blocks that a file can allocate from a single cylinder group.

-m *minfree* This option specifies the minimum percentage of disk space to allocate as free. The default value is 10 percent.

-o *optimization* This option specifies whether you want the file system to minimize time spent allocating blocks, or to minimize the space fragmentation on the disk. If the **-m** option specifies a value of less than 10 percent, then you should request optimization for space. If the **-m** option specifies a value equal to or greater than 10 percent, you should request optimization for time (since fragmentation is less likely).

NOTE

Because the superblock is not kept in the buffer cache, changes made with **tunefs** take effect only when the program is run on unmounted file systems. If you tune the root file system, you must reboot the system.

For more information about the **tunefs** command, refer to the *OSF/1 System and Network Administrator's Reference*.

Managing PFS File Systems 10

Introduction

The Paragon operating system supports a special file system type called *PFS*, for *Parallel File System*. PFS file systems give applications high-speed access to a large amount of disk storage, and are optimized for simultaneous access by multiple nodes. Files in PFS file systems can be very large (up to several terabytes); the exact maximum depends on your system configuration. Access to PFS file systems also uses an I/O technique called *fast path I/O*, which gives superior performance for large I/O operations (64K bytes or more per read or write). A Paragon system can have any number of PFS and non-PFS file systems.

This chapter discusses system administration of PFS file systems. See the *Paragon™ System User's Guide* for user information on PFS file systems and information on the *parallel I/O calls* users can use to access PFS file systems.

Concepts

Every Paragon system has one or more *disk devices* attached to it. Each disk device is either a single hard disk or a *RAID subsystem*. RAID stands for Redundant Array of Inexpensive Disks; in a RAID subsystem, several hard disks are connected together into a unit that appears to the system as a single large disk drive. File blocks stored to a RAID subsystem are distributed, or *striped*, among the disks within it by the RAID controller hardware.

Each disk device is controlled by an *I/O node*: a compute node with an I/O connection. I/O nodes communicate with the other nodes in the system using the node-to-node message-passing network and with the disk drives using a SCSI interface (or other interface). The I/O nodes may or may not also run application processes; you determine this when you create the system's partitions, as described in Chapter 8. Each I/O node can control up to seven disk devices, and the number of I/O nodes is limited only by the number of slots in the system, so the total amount of disk space that could be installed in a Paragon system is a terabyte or more.

The set of disk devices connected to the Paragon system's I/O nodes is divided into *file systems*. A file system can encompass anything from a portion of the space on one disk device to all of the space on several disk devices. A file system is made accessible by *mounting* it to a directory (this requires system administrator privileges). This directory is called the file system's *mount point*. For example, if the file system */dev/io0/rz0f* is mounted on the directory */home* (the directory */home* is the file system's mount point), whenever you write a file in */home* it is stored in the file system */dev/io0/rz0f*.

Each file system has a *type* that describes its internal structure and determines some of the operations that can be performed on it. The supported file system types are:

UFS	UNIX File System, the standard file system type.
NFS	Network File System, a file system type that represents a file system on another computer on the network.
PFS	Parallel File System, a file system type that is optimized for access by parallel processes. This file system type is unique to operating system.

This chapter discusses how PFS file systems work and how you create and manage them. See Chapter 9 for information on disk devices, file systems in general, and UFS and NFS file systems.

Mounting PFS File Systems

PFS file systems are mounted with the **mount** command or with a line in the */etc/fstab* file, the same as UFS and NFS file systems. You use the same syntax and switches for PFS file systems as for UFS file systems; the only difference is the file system type (**pfs**) and the set of options associated with that file system type.

PFS Stripe Attributes

Each PFS file system has some special attributes, called *stripe attributes*, which UFS and NFS file systems do not have:

- A set of *stripe directories*, which are the set of directories across which the data in the PFS file system is distributed. Each stripe directory can be any directory in the Paragon file system tree.

  The number of stripe directories in a PFS file system is referred to as its *stripe factor*. A set of stripe directories can be collected into a list called a *stripe group*; stripe groups are listed in the file */etc/pfstab*.

  Each stripe directory is typically the mount point of a UFS file system, with each UFS file system residing on a different I/O node. Each of these UFS file systems generally should be as large as possible (maximum 2G-1 bytes) and generally should use the largest possible file system block size (maximum 64K bytes).

- A *stripe unit*, which is the number of contiguous bytes of data from a file that the system stores in each stripe directory before storing data in the next stripe directory.

  For best performance, the stripe unit should be an integer multiple of the file system block size of the file systems containing the stripe directories.

- A *stripe attributes partition*, which is a disk partition in which the system stores information about each striped file in the PFS file system. Objects that are not striped, such as directories and symbolic links, are also stored in this partition.

  The stripe attributes partition can be small, since the amount of information stored for each file is typically only a few hundred bytes. It should be formatted as a UFS file system with a file system block size of 8K bytes (the default).

Using the mount Command

The basic syntax for mounting a PFS file system with **mount** is as follows:

```
# mount -t pfs [ -o options ] device directory
```

The parameters are:

options A comma-separated list of the options applying to the mount. No spaces may appear in this list. For a PFS file system, these options include:

stripedirs=*dir*:*dir*:*dir*...
The set of stripe directories to use for this PFS file system. Each *dir* must be an absolute pathname; each specified directory must exist.

stripegroup=*groupname*
The name of a stripe group from the file */etc/pfstab*. The stripe directories associated with this group become the stripe directories for this PFS file system.

stripedirs and **stripegroup** cannot be used together. If neither is used, the default is **stripegroup=all**. See "Using a Stripe Group" on page 10-7 for more information on stripe groups.

stripeunit=*size*
The stripe unit size for this PFS file system. The *size* can be expressed in bytes (*n*), kilobytes (*n***k**), megabytes (*n***m**), or gigabytes (*n***g**). The letter **k**, **m**, or **g** can be uppercase or lowercase.

If **stripeunit** is not used, the default is **stripeunit=64k**.

device The absolute pathname of the device special file for the PFS file system's stripe attributes partition.

directory The absolute pathname of the PFS file system's mount point.

For example, the following **mount** command mounts a PFS file system on the directory */pfs*:

```
# mount -t pfs -o stripeunit=32K,stripedirs=/.sdirs/vol0:
    /.sdirs/vol1:/.sdirs/vol2 /dev/io1/rz0d /pfs
```

This command should be entered all on one line. If you use the backslash (\) line-continuation character, it must not appear within the *options* list.

In this example, the options specify that any file created in */pfs* is to be striped in 32 KB chunks in a round-robin fashion into the three directories */.sdirs/vol0*, */.sdirs/vol1*, and */.sdirs/vol2*. The device */dev/io1/rz0d* is used as the stripe attributes partition.

Using the /etc/fstab File

The syntax for a PFS file system in the */etc/fstab* file is as follows:

```
device directory pfs options fs_freq fs_passno
```

The *device*, *directory*, and *options* are the same as for the **mount** command. The additional parameters are:

fs_freq Used by the **dump** command to determine which file systems need to be backed up.

fs_passno Used by the **fsck** command to determine the order in which file system checks are done at boot time. Should be 4 or more for a PFS file system.

For example, to make the mount operation described by the previous **mount** command occur automatically at boot time, add the following entry to the */etc/fstab* file:

```
/dev/io1/rz0d /pfs pfs rw,stripeunit=32K,stripedirs=
    /.sdirs/vol0:/.sdirs/vol1:/.sdirs/vol2 1 4
```

This entry must appear all on one line in the */etc/fstab* file.

Mounting UFS File Systems for Stripe Directories

The three stripe directories in the previous examples could reside on the same I/O node. However, for the best I/O performance, each stripe directory should be the mount point of a different UFS file system (or a directory in a different UFS file system), and each of these file systems should be controlled by a different I/O node. For instance, you could perform the following mount operations in addition to the PFS mount (note that the default file system type is *ufs*, so the **-t** switch is not necessary):

```
# mount /dev/io3/rz0a /.sdirs/vol0
# mount /dev/io7/rz0a /.sdirs/vol1
# mount /dev/io8/rz0a /.sdirs/vol2
```

This would cause the given disk partitions, owned by I/O nodes 3, 7, and 8, respectively, to be mounted on the directories */.sdirs/vol0*, */.sdirs/vol1*, and */.sdirs/vol2*. Then, after the PFS mount operation is performed, any file created in */pfs* will be striped across I/O nodes 3, 7, and 8.

Similar to the PFS mount, entries can be added to */etc/fstab* to specify that these stripe directory mount operations occur automatically at boot time. The */etc/fstab* entries for this PFS file system would look like this:

```
/dev/io3/rz0a /.sdirs/vol0 ufs rw 1 4
/dev/io7/rz0a /.sdirs/vol1 ufs rw 1 4
/dev/io8/rz0a /.sdirs/vol2 ufs rw 1 4
/dev/io1/rz0d /pfs pfs rw,stripeunit=32K,stripedirs=
    /.sdirs/vol0:/.sdirs/vol1:/.sdirs/vol2 1 5
```

Each of these four entries must appear on a single line in the */etc/fstab* file.

Note that the last field on each line (*fs_passno*) is 4 for each UFS file system and 5 for the PFS file system. This ensures that the UFS file systems are properly **fsck**'d and mounted before the PFS file system is mounted. If the *fs_passno* value for the PFS file system is less than or equal to the *fs_passno* value for its UFS file systems, the PFS file system could appear to be properly mounted even if one or more of its UFS file systems was not. If this occurs, PFS stripe files will be stored in the root file system, which could cause the root file system to fill up.

The */pfs* mount and the */.sdirs* mounts are illustrated in Figure 10-1.

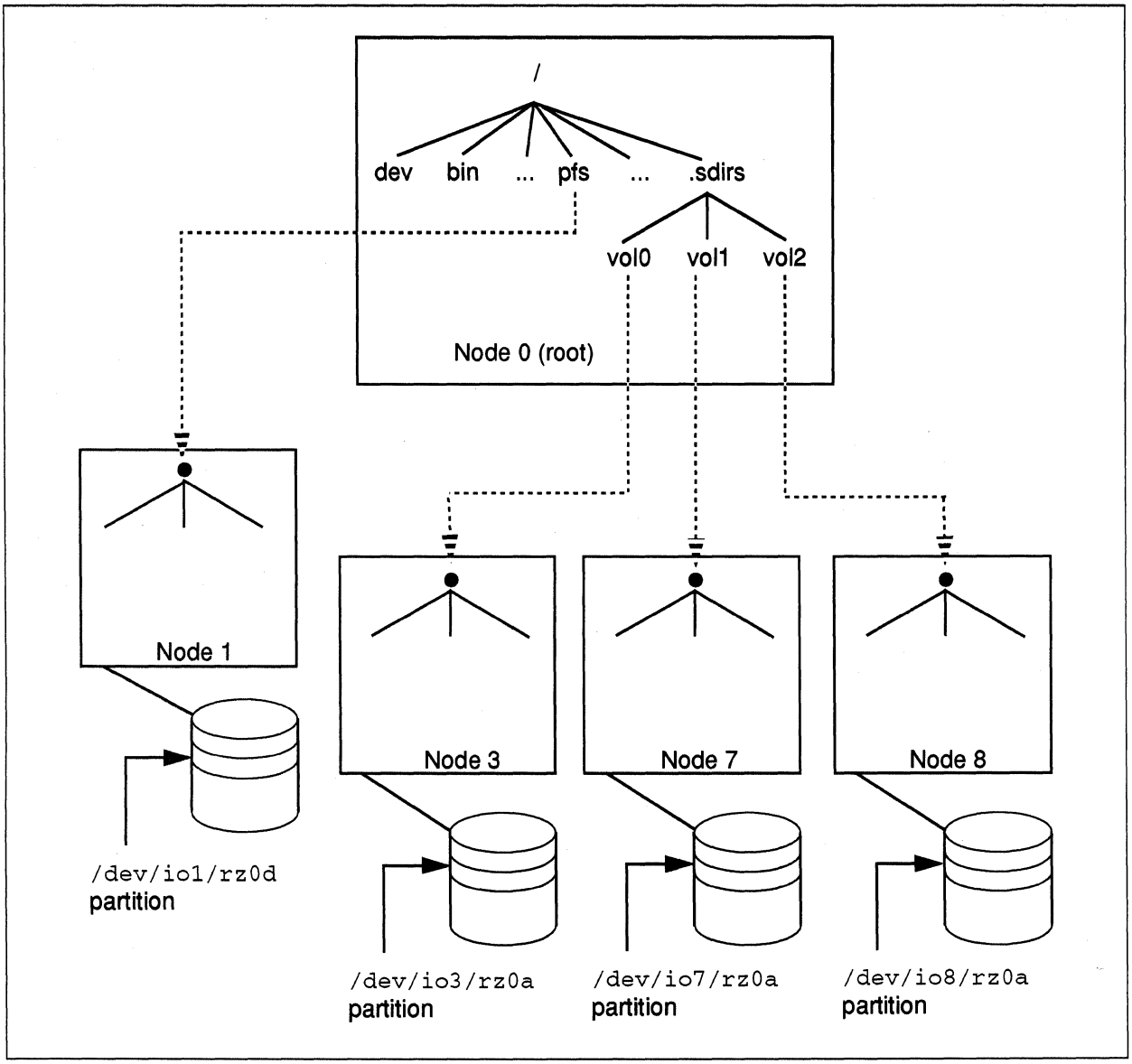


Figure 10-1. Mounting a PFS File System and its Stripe File Systems

Using a Stripe Group

Instead of the **stripedirs** option in the **mount** command or */etc/fstab* file, you can use the **stripegroup** option to specify a group of stripe directories with a single keyword.

The mapping of stripe group keywords to stripe directories is stored in the file */etc/pfstab*. This file contains a list of all the stripe directories in the system. Associated with each stripe directory (on the same line of the file) is a variable number of text fields, each of which contains a stripe group keyword. The presence of a stripe group keyword on a stripe directory's line in */etc/pfstab* indicates that the associated stripe directory belongs to that stripe group.

For instance, on the Paragon system it is common that all I/O nodes are placed on the physical right and left edges of the mesh, and in some cases it may be desirable to stripe PFS files to either one of these sets of I/O nodes. So you might assign stripe directories to stripe groups in */etc/pfstab* as follows (note that all stripe directories always belong to the stripe group *all*):

```
/.sdirs/vol0    all    left
/.sdirs/vol1    all    left
/.sdirs/vol2    all    left
/.sdirs/vol3    all    right
/.sdirs/vol4    all    right
/.sdirs/vol5    all    right
```

Once you had created this */etc/pfstab* file, you could mount a PFS file system that stripes to the I/O nodes on the left-hand side of the system with the following command:

```
# mount -t pfs -o stripegroup=left /dev/io1/rz0d /pfs
```

NOTE

The pound sign (#) character is the comment indicator for both the */etc/fstab* and */etc/pfstab* files.

As a result of this command, **mount** reads the */etc/pfstab* file, extracts the stripe directories in the stripe group *left* in the order in which they occur in the file, and mounts the PFS file system using the ordered list of stripe directories. Then, until the PFS file system is unmounted, any file created in */pfs* will be striped across I/O nodes 3, 7, and 8. In this case the default stripe unit of 64K bytes is used.

Defaults

If any of the stripe options are omitted in the **mount** command or */etc/fstab* file, defaults are used. The **stripeunit** option defaults to 64K bytes, and if neither of the **stripedirs** or **stripegroup** options are specified, then *all* is used as the default **stripegroup**. In other words, the default action is to stripe PFS files across all available stripe directories, using stripe units of size 64 KB. Thus, using the example */etc/pfstab* file in the previous section, the following commands are all equivalent:

```
# mount -t pfs /dev/io1/rz0d /pfs

# mount -t pfs -o stripegroup=all /dev/io1/rz0d /pfs

# mount -t pfs -o stripeunit=64K,stripegroup=all /dev/io1/rz0d
      /pfs

# mount -t pfs -o stripeunit=64K,stripedirs=/.sdirs/vol0:
      /.sdirs/vol1:/.sdirs/vol2:/.sdirs/vol3:/.sdirs/vol4:
      /.sdirs/vol5 /dev/io1/rz0d /pfs
```

Note that each of these commands should be entered on one physical line; line continuation characters (/) cannot be used within the *options* string.

Backing Up and Restoring PFS Files

To backup and restore PFS directories, use the **dump** and **restore** commands, respectively.

NOTE

Since tape devices are typically slow as compared to other storage media and because PFS file systems are usually very large, the following procedure is very time-consuming.

From your Paragon system, enter the following commands to back up your PFS file system:

1. Set the proper environment variable locating the tape device:

```
% setenv TAPEDEV /dev/io0/rmt6
```

2. Load an empty cartridge into the tape deck and back up the main PFS directory:

```
% dump -f $TAPEDEV -c -d 6100 -S 2900 /pfs
```

3. Place an empty cartridge into the tape drive for *each* of the following **dump** commands to back up all PFS stripe directories:

```
% dump -f $TAPEDEV -c -d 6100 -S 2900 /home/.sdir/vol1
% dump -f $TAPEDEV -c -d 6100 -S 2900 /home/.sdir/vol2
.
.
.
% dump -f $TAPEDEV -c -d 6100 -S 2900 /home/.sdir/voln
```

NOTE

If you use a 120-meter cartridge, you can hold up to two gigabytes of data.

In the example above:

-f $TAPEDEV	Specifies the environment variable for the tape drive at */dev/io0/rmt6*.
-c	Specifies a tape device other than a 9-track cartridge tape.
-d 6100	Specifies the write density of 6100 bits per inch (bpi) for the specified tape device. Refer to the documentation of your tape drive for this value.
-S 2900	Specifies an output file size of 2900 feet. This file size will back up a */vol* directory.
/pfs	Specifies the PFS file system.

Enter the following commands to restore your PFS file system:

1. Set the proper environment variable locating the tape device:

```
% setenv TAPEDEV /dev/io0/rmt6
```

2. Change to the PFS directory:

```
% cd /pfs
```

3. Load the correct tape with the files that you wish to restore and enter the following **restore** command:

```
% restore rf $TAPEDEV
```

The **rf** option reads the tape device specified in the environment variable TAPEDEV.

Mounting Multiple PFS File Systems

There's no reason to have just one PFS file system, and it doesn't have to be mounted on */pfs*. You can have any number of PFS file systems at the same time, and you can mount them anywhere in your directory structure.

In fact, you might want to divide your disk nodes into several smaller PFS file systems rather than having one large one. This reduces the maximum size of a file in each file system and may result in lower I/O performance. However, if you distribute your users among the different PFS file systems, you can prevent contention between users for the same file system (thus increasing I/O performance) and increase system availability (if one I/O node or RAID subsystem has hardware problems, users can still get work done by using a different PFS file system).

Managing Stripe Files and Stripe Directories

As already described, a Paragon Parallel File System stores stripe units in regular UFS files inside the stripe directories; these files are referred to as PFS *stripe files*. Stripe files are managed by the Parallel File System, so generally they are "invisible" to the user. However there is nothing to prevent the user from examining them given the appropriate file permissions.

Stripe Attributes Files and Stripe Files

Whenever a user creates an ordinary file in a PFS file system, it is stored as follows:

- The contents of the file are distributed among a series of *stripe files* in the stripe directories.
- Information about the way the file is striped is stored in a *stripe attributes file* in the stripe attributes partition.

When a user creates an object that is not an ordinary file (such as a directory or named pipe) in a PFS file system, the object is stored in the stripe attributes partition only.

Stripe Attributes Files

The stripe attributes file is the only part of the file that is normally visible to the user. As far as the user is concerned, the stripe attributes file *is* the PFS file. In particular, the pathname of the stripe attributes file is the pathname of the user's file within the PFS file system. However, this file does not actually contain the file's data; instead, it contains information about the corresponding stripe files (where the data is actually stored).

The stripe attributes file is treated something like a link; the PFS file system translates any action on the stripe attributes file to the corresponding action on the stripe files. For example:

- A **read**() or **write**() on the stripe attributes file reads or writes the appropriate part(s) of the corresponding stripe file(s).
- A **chmod**() or **chown**() on the stripe attributes file changes the permissions or ownership of the stripe attributes file and all the corresponding stripe files.
- A **stat**() on the stripe attributes file returns information on the stripe attributes file and all the corresponding stripe files, treated as a single file (for example, the returned size is the total of the sizes of all the stripe files).
- An **unlink**() on the stripe attributes file removes the stripe attributes file and all the corresponding stripe files.

In general, users can ignore the stripe files and treat the stripe attributes file as the entire file. However, as the system administrator, you must understand stripe files as well.

Stripe File Names

Each stripe file is stored in the stripe directory itself (never in a subdirectory of the stripe directory) and has the following name:

file_name.*node_num*.*device_num*.*inode_num*.*stripefile_num*

The meaning of each field in the name is as follows:

file_name	Basename of the corresponding stripe attributes file.
node_num	Root partition node number of the I/O node controlling the stripe attributes partition.
device_num	Major and minor device numbers of the stripe attributes partition, encoded as a single value.
inode_num	Inode number of the stripe attributes file.
stripefile_num	Enumeration number of the stripe file (0, 1, 2...). This number makes the stripe file name unique if the PFS file system uses a single stripe directory more than once.

The ..._*num* fields are encoded in an ASCII hexadecimal format.

The *file_name* field in the stripe file name is static: it does not change if the user renames the associated PFS file to a file name within the same PFS file system. This field is only included as a convenience for the system administrator, since generally there is no reason for users to examine the contents of stripe directories. The *node_num*, *device_num*, and *inode_num* together are sufficient to uniquely identify the stripe attributes file associated with the stripe file.

Note that stripe files do *not* reflect the directory structure of the PFS file system. The *file_name* encodes only the *basename* of the corresponding stripe attributes file, not its full pathname, and all stripe files are stored in the same stripe directories. For example, if a user creates two files with the same name in different subdirectories of a single PFS file system, the corresponding stripe files would be in the same stripe directories and would have the same *file_name*, *node_num*, *device_num*, and *stripefile_num* fields. However, their *inode_num* fields would be different.

Stripe File Contents

Each stripe file contains all the parts of the PFS file's data that are stored into the corresponding stripe directory. For example, if a PFS file system is striped across three stripe directories (that is, it has a stripe factor of 3):

- The first stripe file (*stripefile_num* = 0) contains parts 1, 4, 7, 10... of the file's data.
- The second stripe file (*stripefile_num* = 1) contains parts 2, 5, 8, 11... of the file's data.
- The third stripe file (*stripefile_num* = 2) contains parts 3, 6, 9, 12... of the file's data.

The size of each part is the PFS file system's stripe unit. For example, if the PFS file system has a stripe unit of 32K bytes, the first 32K bytes of the file's data would be stored in the first stripe file, the second 32K would be stored in the second stripe file, the third 32K would be stored in the third stripe file, the fourth 32K would be stored in the first stripe file again, and so on in a round-robin fashion.

Example of File Striping

Suppose the PFS file system */pfs* was mounted as shown in the example on page 10-4 (striped across the three directories */.sdirs/vol0*, */.sdirs/vol1*, and */.sdirs/vol2*). Now suppose a user executes the following commands:

```
% mkdir /pfs/mydir
% touch /pfs/mydir/data.out
```

Because directories are not striped, the **mkdir** command creates the directory *mydir* in the root directory of the stripe attributes partition */dev/io1/rz0d*; no stripe files are created at this point.

The **touch** command then might create the following files:

```
/pfs/mydir/data.out                    (stripe attributes file)
/.sdirs/vol0/data.out.7.204.9b8.0      (first stripe file)
/.sdirs/vol1/data.out.7.204.9b8.1      (second stripe file)
/.sdirs/vol2/data.out.7.204.9b8.2      (third stripe file)
```

In this example, the root node number of the I/O node owning the stripe attributes partition (*/dev/io1/rz0d*) is 7, the major/minor device number of the stripe attributes partition itself is 204, and the inode number of the attributes file (*/pfs/mydir/data.out*) is 9b8.

If the user then ran a program that wrote large amounts of data to */pfs/mydir/data.out*:

- The first 32K would be stored in the file */.sdirs/vol0/data.out.7.204.9b8.0*.
- The second 32K would be stored in the file */.sdirs/vol1/data.out.7.204.9b8.*1.
- The third 32K would be stored in the file */.sdirs/vol2/data.out.7.204.9b8.*2.
- The fourth 32K would be stored in the file */.sdirs/vol0/data.out.7.204.9b8.0*.
- The fifth 32K would be stored in the file */.sdirs/vol1/data.out.7.204.9b8.*1.
- ... and so on.

Stripe Directory Permissions

As discussed under "Stripe Attributes Files" on page 10-10, a **chmod()** or **chown()** on a stripe attributes file changes the permissions or ownership of all the stripe files as well as the stripe attributes file. This means that, in most cases, access to stripe files is protected in the same way as access to the corresponding stripe attributes file. For example, you can prevent other people from reading a PFS file by using the command **chmod 700** on the stripe attributes file, as you'd expect.

However, because all stripe files for a given PFS file system are stored in a single set of stripe directories, the permissions of the stripe *directories* have some special considerations. This section discusses those considerations.

Preventing Unwanted Access to Stripe Directories

As described under "Stripe File Names" on page 10-11, stripe files do *not* reflect the directory structure of the PFS file system; all the stripe files for a single stripe directory are stored directly in that directory. This means that permissions of directories within a PFS file system may not protect access to the files in that directory as the user might expect. For example, suppose a user creates a directory and a file in a PFS file system with the following commands:

```
% mkdir /pfs/private
% touch /pfs/private/myfile
% chmod 700 /pfs/private
% ls -ld /pfs/private /pfs/private/myfile
drwx------   2 pat  users  512 May 03 15:17 /pfs/private
-rw-r--r--   1 pat  users    0 May 03 15:17 /pfs/private/myfile
```

In this case, because the directory */pfs/private* is unreadable by others, no other user would be able to read the contents of *myfile* or delete *myfile*. Note, though, that the permissions of *myfile* itself would still allow other users to read it (if they could get to it). If the corresponding stripe directories are readable by other users, those users could access the contents of *myfile* by reading the corresponding stripe files. If the stripe directories are writable by other users, those users could also delete the contents of *myfile* by deleting its stripe files.

To limit the possibility of this type of improper access to PFS stripe files, you should *always* do the following:

- Turn off all *read* permissions on stripe directories, but leave *write* and *execute* permissions for those users that need to create PFS files. This makes it impossible for any user other than *root* to search the directory tree, and prevents any access to a stripe file *unless the full pathname to the file is known.*

- Turn on the *sticky* bit in each stripe directory. With this bit on, a stripe file can only be removed or renamed by a user if the user has write permission for the stripe directory *and* the user is owner of the file, the owner of the stripe directory, or *root*.

For example, to turn off read permissions and set the sticky bit on the stripe directories used in the example on page 10-4, while leaving write and execute permissions on for all users, use the following command:

```
# chmod 1333 /home/.sdirs/vol?
```

The following additional steps can be taken if desired:

- Add a new *pfs* group to the */etc/group* file, to which all users manipulating striped files must belong. Then set the group of all stripe directories to group *pfs* and give only the owner (*root*) and the group (*pfs*) *write* and *execute* permissions on the stripe directories.

- If only one user is using a particular PFS file system, give only that user *write* and *execute* permissions on the stripe directories.
- Begin stripe directory pathnames with '.', so they don't appear in directory listings unless the **-a** switch is used with the **ls** command.

Note that different policies may be used in different PFS file systems.

Group Ownership of Stripe Directories

When a file is created, the owner of the new file is the user who created it, but the group of the new file is the group of the directory containing the file. This can create problems if the group of a stripe directory is not the same as the group of the corresponding directory in the stripe attributes partition.

For example, suppose a user creates a directory and a file in a PFS file system with the following commands:

```
% mkdir /pfs/ourdir
% chgrp new /pfs/ourdir
% touch /pfs/ourdir/ourfile
% chmod 660 /pfs/ourdir/ourfile
% ls -ld /pfs/ourdir /pfs/ourdir/ourfile
drwxr-xr-x   2 pat   new   512 May 03 15:00 /pfs/ourdir
-rw-rw----   1 pat   new     0 May 03 15:00 /pfs/ourdir/ourfile
```

In this case, the stripe attributes file */pfs/ourdir/ourfile* has group *new* and has permissions `rw-rw----`, which make it readable and writable by the owner and by members of that group. The corresponding stripe files also have permissions `rw-rw----`. However, the stripe files have the same group as the stripe directories (for example, *users*). This means that any attempt by members of group *new* other than the user *pat* to read or write */pfs/ourdir/ourfile* will fail (unless they are also members of group *users*).

This problem cannot currently be prevented. However, there are two things you can do to reduce the chance it will occur:

- Be sure that the group of the root directory of the stripe attributes partition is the same as the group of every stripe directory. Never change one without changing them all. This will prevent the problem from occurring unless users use the **chgrp** command on subdirectories of the PFS file system. For best results, use one group (such as *users*) for all stripe attributes partitions and all stripe directories, and be sure that all users belong to this group.
- Inform your users that if members of a group have problems accessing a PFS file, they should use the **chgrp** command to set its group to the expected group. This will have no visible effect on the stripe attributes file, but the explicit **chgrp** will override the stripe files' default group and change their group to the expected group.

Maximum Size of a PFS File

Files in UFS file systems are limited to a size of 2G-1 bytes, as are UFS file systems themselves (since the **newfs** command uses **lseek()**). Because each stripe directory exists in a UFS file system, no stripe file can be larger than 2G-1 bytes. This means that a PFS file system must be striped into more than one UFS file system in order to create a file larger than 2G-1 bytes in size. For example, in order to create a 200G-byte file, a PFS file system must stripe into at least 101 different UFS file systems.

The maximum file size or offset in a PFS file system is limited by the stripe factor (number of stripe directories used) and the maximum size of a stripe file. The largest size of a stripe file is *not* simply a limit of 2G-1 bytes, because only full stripe units are written into stripe files. At the end of the stripe file, there is an unusable partial stripe unit, the size of which is described by the following formula:

EOF_partial = (2G–1) **mod** *stripe_unit_size*

So in general, the maximum size of a stripe file is:

(2G–1) – *EOF_partial*

The exception to this rule is stripe file 0, the stripe file residing in the first stripe directory specified in the mount operation. Since this is the first stripe file, if the last stripe unit of the PFS file falls into this stripe file and if the last stripe unit is a partial that fits into the *EOF_partial*, then this stripe file can be up to 2G-1 bytes in size. Taking this into account, the maximum size of a PFS file is:

(((2G–1) – *EOF_partial*) * *stripe_factor*) + *EOF_partial*

Where *stripe_factor* is equivalent to the number of stripe files.

PFS Performance Hints

This section tells you how to configure a PFS file system for the best performance, based on the number of I/O nodes available and the typical application usage model. However, much of the performance of a PFS file system depends on tuning the applications that use it. See the *Paragon™ System User's Guide* for information on improving the I/O performance of parallel applications.

The Default Configuration

The following discussion assumes familiarity with the */etc/fstab* and */etc/pfstab* files. Refer to the previous sections of this chapter and to the **fstab** and **pfstab** manual pages for more information.

When the operating system operating system is installed, a default PFS file system is configured and mounted as part of the installation process. However, since most Paragon systems are shipped with different I/O configurations, it is rarely the case that this PFS mount is optimal; it is intended as an example only.

The default */etc/fstab* entries that are related to PFS are:

```
#
# Remote filesystems
#
#/dev/io1/rz0c /home/.sdirs/vol0 ufs rw 0 5
#/dev/io2/rz0c /home/.sdirs/vol1 ufs rw 0 5
#/dev/io3/rz0c /home/.sdirs/vol2 ufs rw 0 5
#/dev/io4/rz0c /home/.sdirs/vol3 ufs rw 0 5
#
# Parallel filesystems
#
/dev/io0/rz0d /pfs pfs rw,stripegroup=one 0 3
```

Note that the remote (non-boot-node) mounts are commented out, because the installation assumes a default of only a single I/O node (the boot node).

The default */etc/pfstab* file contains the following entries:

```
/home/.sdirs/vol0 all one two three four five six seven eight
/home/.sdirs/vol1 all     two three four five six seven eight
/home/.sdirs/vol2 all         three four five six seven eight
/home/.sdirs/vol3 all               four five six seven eight
/home/.sdirs/vol4 all                    five six seven eight
/home/.sdirs/vol5 all                         six seven eight
/home/.sdirs/vol6 all                             seven eight
/home/.sdirs/vol7 all                                   eight
```

The result of this default configuration is that when the system is booted to multi-user mode, files written to */pfs* will be striped across stripe group *one*, which contains only the stripe directory */home/.sdirs/vol0*. Since there is no remote file system mounted on this directory, the file data goes to the disk partition that is mounted on */home*. (This is why the installation creates the example *.sdirs* directory in */home*. */home* is the largest file system on the boot node and has the most free space.)

Remember that all of this is configurable: for example the stripe directories can be created anywhere, and the stripe groups in */etc/pfstab* can be redefined with different names.

Although it may seem pointless to "stripe" onto only one I/O node, in fact due to the streamlined disk access provided by PFS, higher performance can be obtained than on a standard UFS file system. This is especially the case if the file size is relatively large (more than a few megabytes).

Adding I/O Nodes to the Default Configuration

This section describes how to add additional I/O nodes to the default configuration (which uses only the boot node). These steps must be performed after the base OS installation has been completed.

1. Create device files and format file systems on all disks, as described under "Configuring Additional RAID Subsystems" on page 11-14. Be sure to use one of the following disklabels in the **disklabel** command when labeling each non-boot node's disks:

   1gbraid3pfs non-boot 4G-byte RAID used for PFS

   1gbraid3pfspg non-boot 4G-byte RAID used for PFS and paging

   raid3pfs non-boot 4.7G-byte RAID used for PFS

   raid3pfspg non-boot 4.7G-byte RAID used for PFS and paging

2. When all device files and file systems have been built, mount the file systems. This can be done by hand, or by uncommenting or adding the appropriate entries in */etc/fstab*. For example, if four I/O nodes are added for PFS file striping, the */etc/fstab* entries described under "The Default Configuration" on page 10-16 might be changed as follows:

```
#
# Remote filesystems
#
/dev/io1/rz0c /home/.sdirs/vol0 ufs rw 0 4
/dev/io2/rz0c /home/.sdirs/vol1 ufs rw 0 4
/dev/io3/rz0c /home/.sdirs/vol2 ufs rw 0 4
/dev/io4/rz0c /home/.sdirs/vol3 ufs rw 0 4
#
# Parallel filesystems
#
/dev/io1/rz0a /pfs pfs rw,stripegroup=four 0 5
```

   Note the following changes:

   - The remote (non-boot-node) mounts have been uncommented and the stripe group of the PFS mount has been changed from *one* to *four*. Now files created in */pfs* will be striped across stripe group *four*, which is defined in */etc/pfstab* to consist of the directories */home/.sdirs/vol0*, */home/.sdirs/vol1*, */home/.sdirs/vol2*, and */home/.sdirs/vol3*. Since each of these directories is the mount point of a UFS file system on a different I/O node, concurrent striping to four I/O nodes is achieved. (Remember that as an alternative to specifying the stripe group, the stripe directories can be explicitly specified in the PFS mount.)

   - The device for the stripe attributes partition has been changed from */dev/io0/rz0d* to */dev/io1/rz0a* in order to reduce the load on the boot node.

- The last field on each line (*fs_passno*) has been changed to 4 for each UFS file system and 5 for the PFS file system (see "Mounting UFS File Systems for Stripe Directories" on page 10-5 for more information).

3. After all file systems have been mounted, set the permissions and ownerships of their root directories appropriately. This only has to be done once.

NOTE

Be sure to perform these steps *while the file systems are mounted.*

When a file system is mounted, the permissions of the mounted-on directory are overridden by the permissions of the root of the file system that has just been mounted. If you perform the following steps before mounting the file systems, you will only set the permissions of the mounted-on directories.

A. Set the permissions of the mount point of the PFS file system (that is, the root directory of the stripe attributes partition) to the appropriate value for any top-level directory at your site. For example:

```
# chmod 1777 /pfs
```

B. Set the permissions of all stripe directories to the appropriate value, as discussed under "Preventing Unwanted Access to Stripe Directories" on page 10-14. For example:

```
# chmod 1333 /home/.sdirs/vol?
```

C. Set the group of the root directory of the stripe attributes partition and all stripe directories to the same value, as discussed under "Group Ownership of Stripe Directories" on page 10-15. For example:

```
# chgrp users /pfs
# chgrp users /home/.sdirs/vol?
```

Your new PFS file system is now configured.

File System Block Size

Generally speaking, the file system block size is the basic unit of transfer between the file system and the media on which the data is stored. As discussed in the *Paragon™ System User's Guide*, an application will always get the highest throughput if the read or write request starts on a file system block boundary and is a multiple of the block size. If these conditions are not met, there is usually a higher overhead imposed in the operating system in order to deal with "partial" blocks, or to correctly align user data.

The file system block size is specified for each partition in the disk label on the RAID. It can also be specified as a command line option to **newfs** when the file system is created; if it is not specified **newfs** extracts the default block size from the disk label. There are several ways to check the block size of an existing file system; the easiest is to use the **dumpfs** command. For example:

```
# dumpfs /dev/io1/rrz0a | grep bsize
```

The largest block size supported is currently 64K bytes. This is the block size recommended for file systems that are used for PFS file striping: the larger the block size, the fewer disk accesses are needed to satisfy a large request.

File systems that are used as stripe attributes partitions should have the default UFS block size of 8K bytes. The files that are stored in these partitions are small, and using a larger block size may result in inefficient use of disk space with no gain in performance.

Stripe Unit Size

The stripe unit size refers to the number of contiguous bytes of data that PFS stores in a stripe directory (usually the mount point of a UFS file system) before moving on to the next stripe directory (UFS file system). For reasons described in the previous section, the optimal stripe unit size is a multiple of the block size of the file systems that PFS uses for stripe data storage. However, the stripe unit size is completely independent of the file system block size, and can be set to any value. The default stripe unit size is 64K bytes.

Consider the */etc/fstab* example from page 10-18, where there are four file systems available for PFS striping:

```
#
# Remote filesystems
#
/dev/io1/rz0c /home/.sdirs/vol0 ufs rw 0 4
/dev/io2/rz0c /home/.sdirs/vol1 ufs rw 0 4
/dev/io3/rz0c /home/.sdirs/vol2 ufs rw 0 4
/dev/io4/rz0c /home/.sdirs/vol3 ufs rw 0 4
#
# Parallel filesystems
#
/dev/io0/rz0d /pfs pfs rw,stripegroup=four 0 5
```

In this configuration, assuming each of the UFS file systems has been created with a block size of 64 K bytes, PFS will stripe exactly one file system block onto each I/O node before moving to the next I/O node (remember that the default stripe unit size is 64K bytes if it is not specified otherwise in the PFS mount). If the PFS mount is changed as follows:

```
/dev/io0/rz0d /pfs pfs rw,stripeunit=128k,stripegroup=four 0 3
```

Then PFS will stripe exactly two file system blocks onto each I/O node before moving to the next I/O node.

You can use the **showfs** or **mount** command to display the stripe unit size of an already-mounted PFS file system. For example:

```
# showfs /pfs
Mounted on   512-blks     avail  capacity  sunit sfactor
/pfs          4128988   3567388     4%     65536       4
    sdirectories: /home/.sdirs/vol0
                  /home/.sdirs/vol1
                  /home/.sdirs/vol2
                  /home/.sdirs/vol3
```

In this case, the **sunit** (stripe unit) is 65536 (64K) bytes.

Changing the Stripe Attributes

You can change the stripe directories and stripe unit size of an existing PFS file system by unmounting the PFS file system and remounting with the new attributes. Files in the PFS file system do not have to be removed before this is done; if the file system is remounted with different attributes, PFS will access an old file according to how the file system was mounted at the time the file was created. This is possible because the information about how a PFS file is striped is stored in the file's stripe attributes file (see "Stripe Attributes Files" on page 10-10). To re-stripe an old file with the new stripe attributes, simply copy it to a new file name.

Rather than remounting an old PFS file system with different attributes, you can instead mount a new PFS file system. This new file system can stripe into the same locations as other PFS file systems if desired. For example, an additional PFS mount can be added to our */etc/fstab* entries:

```
#
# Remote filesystems
#
/dev/io1/rz0c /home/.sdirs/vol0 ufs rw 0 4
/dev/io2/rz0c /home/.sdirs/vol1 ufs rw 0 4
/dev/io3/rz0c /home/.sdirs/vol2 ufs rw 0 4
/dev/io4/rz0c /home/.sdirs/vol3 ufs rw 0 4
#
# Parallel filesystems
#
/dev/io1/rz0a /pfs pfs rw,stripegroup=four 0 5
/dev/io2/rz0a /pfs128 pfs rw,stripeunit=32k,stripegroup=four 0 5
```

This adds a new PFS file system called */pfs128* that has a stripe unit of 32K bytes, instead of the 64K byte stripe unit of */pfs*. Both PFS file systems stripe onto the same I/O nodes. This gives the users the choice of two different stripe units for the same file system; they can choose the one that gives the best performance for their application.

Different PFS file systems can also be configured to stripe onto the same I/O nodes but different file systems, or onto different I/O nodes entirely. For example, if some applications absolutely require using small (less than 64K byte) requests, one PFS file system can stripe into file systems with the standard 64K byte block sizes, while another can stripe into file systems with 8K byte block sizes for better performance on small requests.

Managing I/O Interfaces and Devices 11

This chapter describes how to manage I/O interfaces and devices for the UFS, NFS, and PFS file systems.

Maximum Compute Nodes Per I/O node

Applications should not attempt simultaneous I/O from greater than 32 compute nodes to any one I/O node (MIO) at one time. This is true for UFS, NFS, and PFS (striped on one I/O node). This is due to underlying system limitations that cause low bandwidth and/or system hangs when multiple clients are accessing that same I/O node. The system administrator must be careful to configure the PFS file systems so that this restriction is enforced across multiple simultaneous applications. For more information about how to configure PFS systems, refer to "Managing PFS File Systems" on page 10-1. Users must understand PFS I/O modes and stripe attributes in order to code their applications within this restriction. For more information about programming PFS applications, see the *Paragon™ System User's Guide*.

I/O Request Size Limitation

Applications should avoid simultaneous large I/O requests to any single I/O node (MIO). This may cause system performance to degrade and may cause the system to hang. This is due to excessive paging when the total size of the I/O transfers exceeds the physical memory size of the I/O node. A suggested workaround is to break large I/O requests into smaller I/O requests, so the smaller I/O requests will fit into physical memory.

For example, if a 64-node application issues simultaneous requests for 512K-byte I/O from each compute node using the **M_RECORD** I/O mode, a system with two I/O nodes would need up to 16M bytes of buffer space for each I/O node. A system with 16M-byte nodes has approximately 10M bytes available memory, while a system with 32M-byte I/O nodes has approximately 26M bytes available.

Configuring I/O for UFS and NFS Systems

For general information affecting UFS and NFS file systems, refer to "Managing UFS and NFS File Systems" on page 9-1

The default I/O mode that the shared files are open with depends on the type of file and the value of the `PFS_ASYNC_DFLT` bootmagic string. Behavior is as follows:

The default I/O mode is `M_UNIX` for all non-PFS files. This behavior holds true regardless of the `PFS_ASYNC_DFLT` bootmagic string. For PFS files, the default I/O mode is `M_UNIX` when `PFS_ASYNC_DFLT` is set to any value other than 1. When `PFS_ASYNC_DFLT` is set to 1, the default I/O mode is `M_ASYNC`.

For instructions on how to change bootmagic strings, refer to "Changing Default MAGIC.MASTER Strings" on page 4-20.

Configuring an I/O Partition

You can place any I/O node other than the boot node in the service partition, in the compute partition, or in a separate I/O partition. The advantage of a separate I/O partition is that you can restrict its permissions so that no user processes can run on those nodes. Doing this can increase I/O efficiency and make I/O performance more consistent by eliminating all load on the I/O nodes other than system I/O processes. The disadvantage of a separate I/O partition is that those nodes cannot be used for other work—when there is no I/O, they sit completely idle and do not contribute to the performance of the system.

Use the following rules when you configure your system with an I/O partition.

- Use standard scheduling.
- I/O partition should be owned by user *root* and group *daemon*
- Set permissions to 754 (`rwxr-xr--`).

  Note that user I/O processes are not controlled by the allocator, so they are not affected by the partition's permissions.
- Do not create subpartitions

Understanding the */dev* Directory

The */dev* directory contains device (or "special") files. Device files are of particular importance within the system environment since they serve as the link between the system and the device drivers. Each device file corresponds to a physical device (a disk, tape, printer, or terminal, for example) or a pseudo device (a network interface, a named pipe, or domain socket, for example). The driver knows how to handle all read and write operations that relate to the specific device, and will follow the required protocols for that device.

In the Paragon system, some physical devices are attached to *remote nodes* (that is, I/O nodes other than the boot node). The device files under */dev* contain the information the system needs to find the device, no matter where it is in the system. You can see the node number of a file's associated device with the command **ls -l**, as shown in the following listings.

There are three types of device files:

- Block device files are used for devices where the driver handles I/O in large blocks and where the operating system handles I/O buffering. Many physical devices such as disks and tapes are defined as block device files. Block device files are listed in the */dev* directory with notation like the following:

```
brw-r--r--   1 root  system   7:  3,  0 Apr 20 16:36 rz0a
brw-r--r--   1 root  system   7:  3,  1 Apr 20 16:36 rz0b
brw-r--r--   1 root  system   7:  3,  2 Apr 20 16:36 rz0c
brw-r--r--   1 root  system   7:  3,  3 Apr 20 16:36 rz0d
```

  The `b` at the beginning of the line identifies the file as a block device file, and the three numbers shown where the size would normally appear identify the physical device associated with the file:

  - The first number (in this case 7) is the *root node number* of the node to which the corresponding physical device is attached.

  - The second number (in this case 3) is the *major device number* of the device, which indicates the device driver software used to access the device. The association of major device numbers with device drivers is determined by a table within the operating system.

  - The third number (in this case 0, 1, 2, or 3) is the *minor device number* of the device. The meaning of the minor device number is determined by the device driver. For example, for a disk device the minor device number indicates the partition on the disk; for a tape device the minor device number indicates how the tape is used (rewinding or non-rewinding).

- Character device files are used for devices where the driver handles its own I/O buffering. Terminal and pseudo-terminal drivers are typically associated with character device files. Character device files are listed in the */dev* directory with notation like the following:

```
crw-rw-rw-   1 root  system   7: 10,  0 Apr 27 16:26 ptyp0
crw-rw-rw-   1 root  system   7: 10,  1 Apr 25 15:45 ptyp1
crw-rw-rw-   1 root  system   7: 10,  2 Apr 25 15:45 ptyp2
crw-rw-rw-   1 root  system   7: 10,  3 Apr 20 16:42 ptyp3
crw-rw-rw-   1 root  system   7: 10,  4 Apr 20 16:42 ptyp4
```

  The `c` at the beginning of the line identifies the file as a character device file. The three numbers shown where the size would normally appear identify the node number and major and minor device number of the corresponding physical device, as discussed above for block device files.

- Socket device files are used by the printer and error logging daemons. Socket device files are listed in the */dev* directory with notation like the following:

```
srw-rw-rw-   1 root  system          0 Apr 20 16:42 log
srwxrwxrwx   1 root  system          0 Apr 20 16:42 printer
```

  The `s` at the beginning of the line identifies the file as a socket device file. Socket device files are not associated with physical devices, so they don't have node numbers or major or minor device numbers. The size of a socket device file is always 0.

Refer to "Creating Device Files" on page 11-5 for instructions on using the **mknod** command to create device files and more information on major and minor device numbers.

Subdirectories of */dev*

In addition to device files, the */dev* directory contains one directory for each I/O node in the system. Each directory has the name */dev/ion*, and contains the disk and tape device files for the *n*th I/O node in the system (numbered sequentially from 0). Thus, the directory */dev/io0* contains the device files for the disk and tape devices connected to the boot node; the directory */dev/io1* contains the device files for the disk and tape devices connected to the first I/O node after the boot node, and so on. To determine the root node number of the node represented by each of these directories, use the command **ls -l** on any of the files in the directory.

Some I/O nodes may have two SCSI channels with devices attached to both. Devices on the second SCSI channel are in the */dev/ion/ch1* directory.

Creating Device Files

Device (special) files correspond to physical devices, pseudo devices, and named pipes. To automatically create the */dev/ion* directories and device files for all SCSI disk and tape devices on all I/O nodes in the system, use the **MAKEDEV** script. This script, which is located in the */dev* directory, performs the following steps:

- Finds the list of all the available I/O nodes in the system and creates a directory for each I/O node under the */dev* directory. (See "Subdirectories of /dev" on page 11-4 for more information on these directories.)
- Checks all the SCSI IDs for each of the I/O nodes in the system.
- Determines what type of devices (disk or tape) are attached to each node.
- Checks that a device file exists for each SCSI device. Creates any needed device files that do not exist.
- Finds any device files that do not correspond to actual devices and asks if you want to remove them. If you answer "y," removes them.
- Finds any RAID level 5 devices and asks if you want to convert them to RAID level 3. If you answer "y," converts them.

You can also specify a single root node number, to create device files for just that node instead of all nodes in the system.

To create all disk and tape device files for all nodes in the system, or a specified node:

1. As root, change directory to */dev*:

```
cd /dev
```

2. Create device files with the following command syntax:

```
./MAKEDEV [node]
```

For more information about the **MAKEDEV** command, refer to the *Paragon™ System Commands Reference Manual.*

Creating a Boot Node Device File or Named Pipe

To create a single device file for a device connected to the boot node, or a named pipe, follow this sequence:

1. As root, change directory to */dev*. For example, enter:

```
cd /dev
```

2. Create a device file with the following command syntax:

```
mknod device_name [device_type major# minor#]
```

The arguments are described as follows:

device_name Specifies the name of the file that will represent the device. When naming the device, follow the conventions and mnemonics of the current (default) directory entries. In any case, use a device mnemonic that you can easily recognize as corresponding to the device. For example, the *tty* mnemonic is easily recognized as representing a terminal, *pty* corresponds intuitively to a pseudo terminal device. It is useful to distinguish between block and character device files for devices that handle both types. For example, you might use *rrz* (the raw device mnemonic) for character device files and *rzz* for block device files on SCSI disks.

device_type Specifies whether the device is a block, character, or named pipe device file. Use **b** to specify a block device file, **c** to specify a character device file, or **p** to specify a named pipe device file.

major# Specifies the major device number associated with the device type. A major device number is an index to a table within the operating system. If you have source for the operating system, see the *conf.c* source file for a listing of device to major device number associations; otherwise, contact SSD Customer Support.

minor# Specifies the minor device number associated with the device type. The meaning of each minor device number is determined by the device driver. If you have source for the operating system, see the source of the device driver for information on the minor device numbers supported by the driver; otherwise, contact SSD Customer Support.

Creating a Remote Node Device File

If you want to create a single device file associated with a physical block or character device on a node other than the boot node, you must use the **rmknod** command:

1. As root, change directory to the */dev/ion* directory corresponding to the node to which the device is attached (see "Subdirectories of /dev" on page 11-4 for more information). For example, to create a device for the third I/O node in the system (counting the boot node):

   ```
   cd /dev/io2
   ```

   If the specified directory does not exist, you must create it.

2. Create the device file with the **rmknod** command.

   For a specific example, see step 6 of "Major and Minor Numbers for Devices" on page 11-7. For more information about the **rmknod** command, refer to the Manpages or the *Paragon™ System Commands Reference Manual.*

Relation of Minor Device Numbers to Physical devices

The minor device numbers have a direct relationship to the device hardware. Figure 11-1 on page 11-8 shows an example bit representation of one of two drive units (Logical Unit Number 0) of a 3480 Tape drive. The tape drive resides on a SCSI bus. A MIO controller on the node board controls the tape drive (Target ID 6) along with all other devices on the bus. If you were to enter the following list command in the */dev/io0* directory,

```
ls -l
```

you would see the major and minor numbers for all I/O devices in */dev/io0*. The example below shows the device entry for the 3480 tape drive with a minor number of 96 (Decimal).

```
crw-rw-rw-   1 root      10          3:  8, 96 Jul 06 10:36 rmt6
```

Decimal 96 in binary format is `0b01100000`, representing bits in the device controller. The figure shows that the bits select the MIO controller (controller 0), the 3480 tape unit itself (Target ID 6), the No-rewind on Close feature, and Logical Unit Number 0 (LUN0).

Major and Minor Numbers for Devices

Table 11-1 on page 11-9 shows a standard device names and their associated major and minor device numbers.

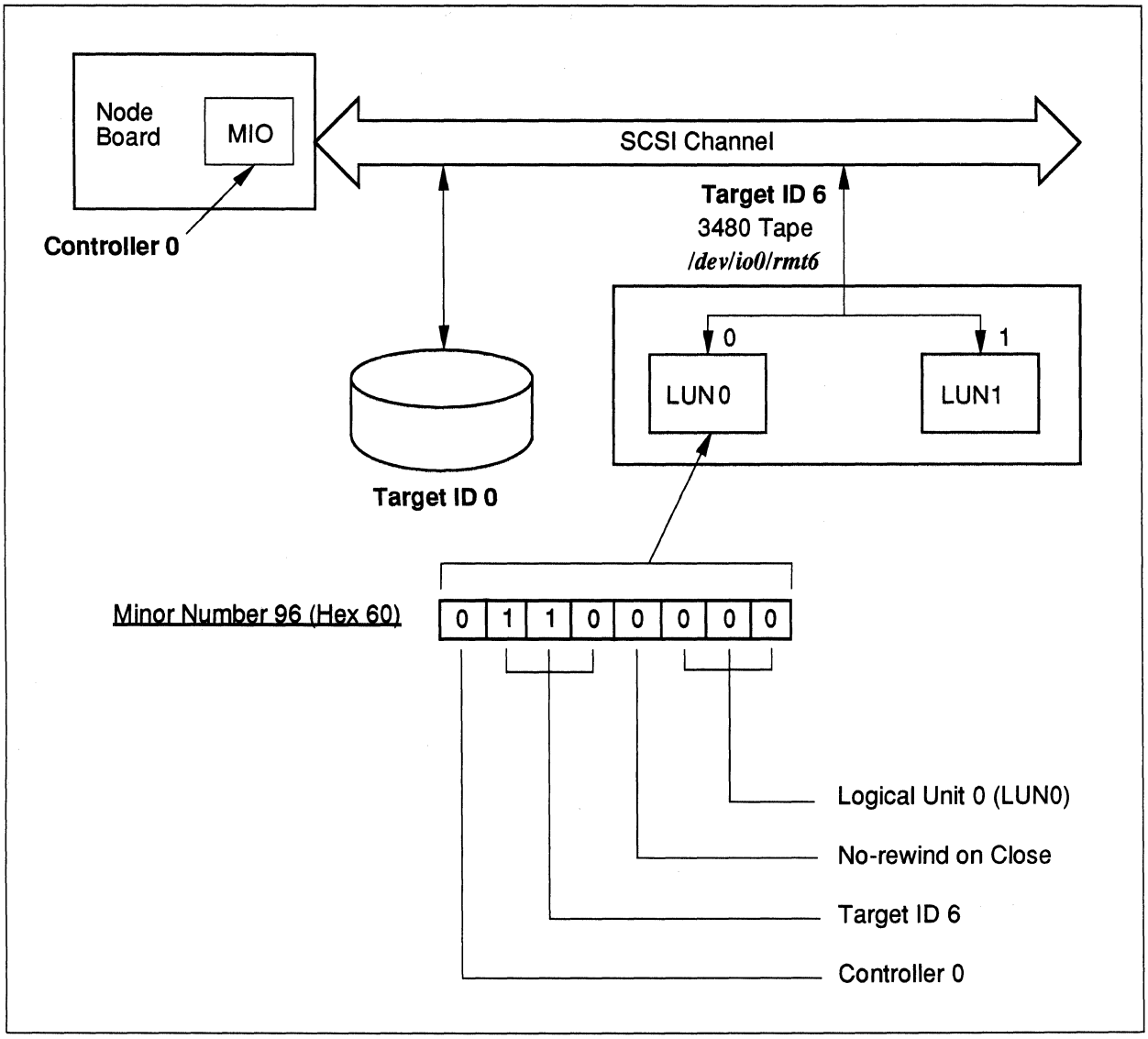


Figure 11-1. Relationship of Minor Device Number to I/O Hardware (tape id 6 lun 0)

Table 11-1. Device Names and Major/Minor Device Numbers (1 of 2)

Device Name	Description	Type *Note* 1	Major Number	Minor Number	Max. Units
console	Console terminal	c	1	Always 0	1
md*(0a - 0d)*	Memory device used during software installation (RAM disk).	b	7	0 for first device, 1 for second device, etc.	4
null	Special device file for data sink	c	3	Always 2	1
pty*(p0 - uf*	Pseudo-terminal device	c	10	0 for first device, 1 for second device, etc.	94
rmd*(0a -0d)*	Raw device for MD	c	22	0 for first device, 1 for second device, etc.	4
tty	Terminal device	c	2	Always 0	1
tty*(p0 - uf*	Terminal device	c	9	0 for first device, 1 for second device, etc.	94
nrmt*6*	No-rewind on Close tape device	c	8	<channel> * 128 + <integer *y*> * 16 + <offset *z*> integer *y* = 6 for first device, 5 for next device, etc. offset z (no-rewind tape) = 8	1 *Note* 2
nrmtc*6*	No-rewind on Close tape device with compression	c	7	<channel> * 128 + <integer *y*> * 16 + <offset *z*> integer *y* = 6 for first device, 5 for next device, etc. offset z (no-rewind tape) = 8	1 *Note* 2
rmt*6*	Rewind on Close tape device	c	8	<channel> * 128 + <integer *y*> * 16 + <offset *z*> integer *y* = 6 for first device, 5 for next device, etc. offset z (rewind tape) = 0	1 *Note* 2
rmtc*6*	Rewind on Close tape device with compression	c	7	<channel> * 128 + <integer *y*> * 16 + <offset *z*> integer *y* = 6 for first device, 5 for next device, etc. offset z (rewind tape) = 0	1 *Note* 2
rrz*(0a - 0h)*	Raw disk device	c	23	<channel> * 128 + <integer *y*> * 16 + <partition> First partition is always 0, 1 for next partition, etc.	*Note* 2

Table 11-1. Device Names and Major/Minor Device Numbers (2 of 2)

Device Name	Description	Type *Note* 1	Major Number	Minor Number	Max. Units
rz*(0a - 0h)*	Block disk device	b	3	<channel> * 128 + <integer *y*> * 16 + <partition> First partition is always 0, 1 for next partition, etc.	*Note* 2
rimd(0a-0h)	Raw disk device (IPI-3 device)	c	33 + *slave_id*	First partition is always 0, 1 for next partition, etc.	7 *Note* 3
ripm(0a-0h)	Raw master device (IPI-3 device)	c	25 + *slave_id*	Does not use the minor number	1 *Note* 3
imd(0a-0h)	Block disk device (IPI-3 device)	b	23 + *slave_id*	First partition is always 0, 1 for next partition, etc.	7 *Note* 3
scsi*(0 or 6)*	SCSI pass-through device	c	23	<channel> * 128 + <integer *y*> * 16 + <offset *z*> integer *y* = 0 for first device integer *y* = 6 for second device offset *z* = 15	2

1. In the Type column, the argument *b* indicates block-oriented devices and *c* indicates character-oriented devices.

2. One SCSI bus includes a combination of up to eight tape or disk devices (and the SCSI controller itself). Disk devices are numbered from 0 to 6; tape devices are numbered from 6 to 0.

3. The **MAKEDEV** command does not handle IPI-3 devices. You must calculate the major and minor device numbers using this table. Refer to the *Paragon™ System High Performance Parallel Interface Manual* to calculate *slave_id* information.

You use major and minor device numbers when creating special devices files to the */dev* directory. You will need the table above to calculate the major and minor device numbers that the **rmknod** command needs to create these special device files.

When your system was installed, the **MAKEDEV** program assigned all of these numbers automatically. On occasion you will need to assign new device names with their associated major and minor numbers without using the **MAKEDEV** program. See the manpage for the **MAKEDEV** command for more information.

For example, if you have physically installed a rewind on close tape drive (with compression) and you want to place the device in the */dev/io* directory without running **MAKEDEV**, you would perform the following steps:

1. Find the standard name for a rewind on close tape device (with compression).

   Using Table 11-1, you would note the standard device name for that for that tape device (in this case, the name is *rmtc6*). For now, write this name on a piece of paper.

2. Find the major device number for this type of device.

   Table 11-1 lists the major device number as 7 for this device.

3. Calculate the minor device number.

   Using the formula in Table 11-1, note the following values:

   ```
   integer y = 6
   offset z = 0
   ```

   Substituting these table values into the formula, you can calculate the minor device number:

   ```
   6 * 16 + 0 = 96
   ```

4. From the type column, note if the device is character-oriented or block oriented type.

   This device is a block-oriented (*c*) type device.

5. Write down the node number that you have physically installed the device.

   For this example, assume you have installed the device on node 2.

6. Enter the **rmknod** command using the device name, type of device, major number, minor number, and node number:

   ```
   rmknod /dev/io/rmtc6 c 7 96 2
   ```

   This command places the special file *rmtc6* into the */dev/io* directory as a character-oriented device with 7 as the major device number and 96 as the minor device number on node 2.

Configuring an Additional Ethernet Board

Every Paragon system has at least two Ethernet connections, one to the diagnostic station and one to the boot node. Follow these instructions if you have another Ethernet connection to an I/O node other than the boot node. This makes for a total of three Ethernet connections, but because this is a second Ethernet connection to the Paragon nodes, these instructions are often referred to as those for configuring a second Ethernet.

1. Log in or **rlogin** to the diagnostic station as *root*.

2. Change directory to */usr/paragon/boot*. Edit *DEVCONF.TXT* to contain a line identifying the position of the second Ethernet connection.

   For example, consider a two-cabinet configuration with one Ethernet connection. This connection is on the boot node which is in cabinet 0, backplane D and slot 3. The CBS number of the boot node is 00D03. Now assume that you add a second Ethernet connection to a new I/O node in cabinet 1, backplane A, and slot 12. Its CBS number is 01A12.

   A typical *DEVCONF.TXT* file is shown below. The lines in **bold** are those added for the new I/O node with the Ethernet connection.

```
DEVICES
ENET 00D03
ENET 01A12
RAID 00D03 ID 0 SW 3.02 LV 3 DC 5 SID 0 RAID3
TAPE 00D03 ID 6 DAT
MIO 00D03 H04
MIO 01A12 H04
END_DEVICES
```

3. Connect to the Paragon system with the **console** command.

```
DS# cd /usr/paragon/boot
DS# ./console
          .
          .
          .
login:
```

4. Edit the */etc/hosts* file to include the new interface, using the additional IP address and additional host name.

5. Edit the file */etc/rc.config*. Define *HOSTNAME2*, *HOSTID2* and *NETDEV2* and export their values.

```
HOSTNAME=joe
HOSTNAME2=frank
HOSTID=xxx.yy.zz.aaa
HOSTID2=xxx.yy.zz.bbb
NETMASK=255.255.255.00
NETDEV="<7>em0"
NETDEV2="<96>em0"
          .
          .
          .
export DISPLAYTYPE HOSTNAME HOSTNAME2 HOSTID HOSTID2 NETMASK
NETDEV NETDEV2 TZ
```

The *NETDEV* variable identifies the node number of the Ethernet connection. Each Ethernet connection has its own *HOSTNAME* and *HOSTID*. It is not necessary to define a *NETMASK2* if the second Ethernet connection is on the same class as the primary and uses the same mask value.

6. Log in as *root* to the Paragon system and edit the file */sbin/init.d/inet*. The lines in **bold** indicate the lines that must be added to configure a second Ethernet board. Both are **ifconfig** lines.

```
if [ "$ARGTWO" = "force" -o "$9" = "S" ]; then
 echo "Configuring network"
 echo "hostname: \c"
 /sbin/hostname $HOSTNAME
 /sbin/hostid $HOSTID
 echo "hostid: $HOSTID"
 /sbin/ifconfig $NETDEV $HOSTID netmask $NETMASK -trailers
# ifconfig second ethernet
 /sbin/ifconfig $NETDEV2 $HOSTID2 netmask $NETMASK -trailers
 /sbin/ifconfig lo0 127.0.0.1
#
# Add default routes; start routed
#
#                echo "Adding route for osgw: "
#                SUBNET='expr $HOSTID : ".*.*.*<.*>.*"'
#               /sbin/route add default 137.27.$SUBNET.1
                /sbin/route add default $GATEWAY
fi
;;
'stop')
        /sbin/ifconfig $NETDEV2 down
        /sbin/ifconfig $NETDEV down
        /sbin/ifconfig lo0 down
```

7. Unmount all file systems listed in */etc/fstab*, and use the **halt** command to shut down the Paragon system. Then, return to the diagnostic station prompt by typing **~.** (or **~~.** if you are logged in remotely to the diagnostic station). The key sequence **~q** also works and does not require that you keep track of the number of remote logins. Then, issue **reset** with the **autocfg** option. This ensures that the new information in *DEVCONF.TXT* is added to *SYSCONFIG.TXT*.

```
# cd /
# sync;sync;sync
# shutdown now
# umount -A
# halt
# ~.
DS# ./reset autocfg
DS#
```

8. Reset the Paragon system and enter multiuser mode. When the **reset** command has completed and the prompt returns, enter multiuser mode by **<Ctrl-D>**. Note that the **reset** script automatically enters multiuser mode if the *MAGIC.MASTER* file contains RB_MULTIUSER=1 .

```
DS# ./reset
          .
          .
          .
# <Ctrl-D>
```

Configuring Additional RAID Subsystems

This section describes the configuration steps that are necessary if your Paragon has I/O nodes in addition to the boot node that have a RAID subsystem attached.

1. Log in to the diagnostic station as *root*. Determine the CBS (Cabinet:Backplane:Slot) numbers for your I/O nodes, and then check the *SYSCONFIG.TXT* file in the directory */usr/paragon/boot* for correctness.

   For example, consider a configuration that has three I/O nodes, one of which is the boot node. The I/O nodes are located as follows.

   - The boot node has an OS number of 7 and is located in cabinet 0, backplane D, and slot 3. Its CBS number is 00D03.
   - The second MIO node as an OS number of 15 and a CBS number of 00D02. This is an additional I/O node.
   - The third MIO node has an OS number of 23 and a CBS number of 00D01. This is an additional I/O node.

The appropriate portion of a correct *SYSCONFIG.TXT* for this example is shown below. Note that the lines for slots 1, 2, and 3 contain the keyword RAID3 and MIO. Also note that slot 3, which contains the boot node, also has an Ethernet connection identified by the keyword ENET.

```
CABINET 0
PC AU02
LED AM00
BP D AC02
S 0 GPNODE AK00 16 MRC 04
S 1 GPNODE AK00 16 MRC 04 MIO H04 RAID3
S 2 GPNODE AK00 16 MRC 04 MIO H04 RAID3
S 3 GPNODE AN00 32 MRC 04 ENETMIO H04 RAID3 DAT
S 4 GPNODE AK00 16 MRC 04
.
.
.
```

If *SYSCONFIG.TXT* is not correct, then you must edit *DEVCONF.TXT*.

2. Add lines for the additional I/O nodes and RAID subsystems. The additional lines are shown in **bold**.

```
DEVICES
ENET 00D03
RAID 00D03 ID 0 SW 3.02 LV 3 DC 5 SID 0 RAID3
RAID 00D02 ID 0 SW 3.02 LV 3 DC 5 SID 0 RAID3 NOPAGER
RAID 00D01 ID 0 SW 3.02 LV 3 DC 5 SID 0 RAID3 NOPAGER
TAPE 00D03 ID 6 DAT
MIO 00D03 H04
MIO 00D02 H04
MIO 00D01 H04
END_DEVICES
```

Whether or not these new I/O nodes are to be used in the paging tree, identify them as NOPAGER in *DEVCONF.TXT*. This is because until their devices are built with **MAKEDEV** and until their disklabels are written, you must ensure that they are not built into a paging tree.

3. Connect to the Paragon system with the **console** command.

```
DS# cd /usr/paragon/boot
DS# ./console
          .
          .
          .
login:
```

4. Log in as root to the Paragon system, unmount all file systems, and use the **halt** command to shut down the Paragon system. Then, return to the diagnostic station prompt by typing **~.** (or **~~.** if you are logged in remotely to the diagnostic station). The key sequence **~q** also works and does not require that you keep track of the number of remote logins.Then, issue **reset** with the **autocfg** option. This ensures that the new information in *DEVCONF.TXT* is added to *SYSCONFIG.TXT*.

```
# cd /
# sync;sync;sync
# shutdown now
# umount -A
# halt
# ~.
DS# ./reset autocfg
DS#
```

5. Check the */usr/paragon/boot/bootmagic* file to make sure the newly configured I/O nodes have been added to the BOOT_DISK_NODE_LIST line.

```
BOOT_DISK_NODE_LIST=7,15,23
```

6. Reset the Paragon system and enter multiuser mode. When the **reset** command has completed and the prompt returns, enter multiuser mode by pressing **<Ctrl-D>**. Note that the **reset** script automatically enters multiuser mode if the *MAGIC.MASTER* file contains RB_MULTIUSER=1 .

```
DS# ./reset
          .
          .
          .
# <Ctrl-D>
```

7. Check the file */etc/devtab* on the Paragon system. This file must specify the boot node in two different ways: as an OS number and as a CBS (Cabinet-Backplane-Slot) number. The separators on the */dev/io0* line are tabs.

   Verify that */etc/devtab* does not have lines for any devices other than */dev/io0* (such as */dev/io1*, */dev/io2*). If it does, delete the extra lines and set IONODES = 1. The following example shows the correct format:

```
IONODES = 1
/dev/io0      7      0D3
```

8. Change to the */dev* directory. If */dev* contains any *io** directories other than *io0* (such as *io1*, *io2*), remove them. Do not delete */dev/io0*.

```
# cd /dev
```

9. Now, run **MAKEDEV**:

```
# ./MAKEDEV
```

The **MAKEDEV** script creates the device special files */dev/io1*, */dev/io2*, and so on. Devices for the boot node are under */dev/io0* which is made automatically during installation of the system software. Then **MAKEDEV** uses the BOOT_DISK_NODE_LIST to check on the RAID level for each I/O node in the list. If it finds any RAID5 drives, it displays the following messages:

```
The RAID drives are either configured as RAID5, or are
otherwise in need of initialization. This procedure will
destroy any data currently on the drive. Do you want to
continue? (y/n)[n]
ARE YOU SURE? (y/n)[n]
```

If you enter '**Y**' or '**y**' to both questions, the drive is initialized and converted to RAID3. If you do not convert a RAID5 drive to RAID3, the Paragon will not be able to access it. Note that it takes about 20 minutes to format a RAID subsystem.

10. Use the **disklabel** command to label each of the drives installed by **MAKEDEV**. For example, the following line labels the disk using the default boot-node, RAID3 disk label information from the */etc/disktab* file:

```
# /usr/sbin/disklabel -rw /dev/io?/rz0a raid3
```

where *?* is the number of the I/O node (1, 2, 3, ...) in the */dev/io?* directories. *raid3* is the disklabel.

However, this boot node disklabel specifies that all file systems be made with only 8K-byte file system blocks. This is acceptable for UNIX OS binaries and program development, but is less than ideal if you intend to run typical supercomputer applications that perform large I/O operations with PFS.

If the RAID subsystem you are labeling is on a non-boot node, choose another default label. Look in the */etc/disktab* file for provided labels. Some choices are as follows:

4gbraid3pfs	non-boot 4G-byte RAID used for PFS
4gbraid3pfspg	non-boot 4G-byte RAID used for PFS and paging
raid3pfs	non-boot 4.7G-byte RAID used for PFS
raid3pfspg	non-boot 4.7G-byte RAID used for PFS and paging

If none of the provided disklabels fit your needs, you must customize your disklabel as follows:

A. Use the **disklabel** command to read the default information into a file. The following command reads the disklabel from */dev/io1/rrz0a* which in this example turns out to be raid3pfspg.

```
# /usr/sbin/disklabel -r /dev/io1/rrz0a > disklabelfile
```

B. The following listing shows the label for raid3pfspg. This example shows four partitions: *a*, *b*, *c*, and *d*. *a* is a 20M-byte partition used for the PFS mount point. The *a* partition is used for the PFS mount point. The *b* partition is used for paging because it has an 8K-byte block size. The *c* and *d* partitions have 64K-byte block sizes and hence are used for PFS striping.

```
# /dev/io1/rrz0a:
type: SCSI
disk: raid3pfspg
label:
flags:
bytes/sector: 2048
sectors/track: 65
tracks/cylinder: 15
sectors/cylinder: 975
cylinders: 2480
rpm: 6300
interleave: 1
trackskew: 0
cylinderskew: 0
headswitch: 0           # milliseconds
track-to-track seek: 0  # milliseconds
drivedata: 0

4 partitions:
#        size   offset    fstype   [fsize bsize   cpg]
  a:    10240        0    4.2BSD     2048  8192    16   # (Cyl.    0 - 10*)
  b:  1048576    10240    4.2BSD     2048  8192    16   # (Cyl.   10*- 1085*)
  c:  1048576  1058816    4.2BSD     8192 65536    32   # (Cyl. 1085*- 2161*)
  d:   309248  2107392    4.2BSD     8192 65536    32   # (Cyl. 2161*- 2478*)
```

C. Use **disklabel** again to specify the new disk label based on the information from the file.

```
# /usr/bin/disklabel -rR /dev/io1/rrz0a disklabelfile
```

11. After the drive is labeled, use **newfs** to create the file systems on the drive:

```
# newfs  -c 32 /dev/io?/rz0?
```

In this example, replace the *?* in */dev/io?* with the number of the I/O device (*1*, *2*, *3*, etc.), and replace the *?* in *rrz0?* with the partition name (in the example above, either *a*, *b*, *c*, or *d*). This means that the **newfs** command must be repeated for each partition on each device. The following script creates the file systems for partitions *a*, *b*, and *c* on a specified drive. If you try to **newfs** a non-existent partition, you get an error message.

```
#!/bin/sh
if [ $# -lt 1 ]
then
      echo "Usage: $0 <I/O node specification, e.g. io1>"
      exit 1
fi

partitions="a b c"

for part in $partitions
do
      echo "newfs -c 32 /dev/$1/rz0${part}"
      newfs -c 32 /dev/$1/rz0${part}
      echo ""
done
```

12. If the new I/O nodes are to be used in a paging tree, edit *DEVCONF.TXT* and remove the NOPAGER tokens. Then, issue a **reset autocfg** followed by a **reset.**

13. The partitions are now ready for mounting. To mount a partition create a mount point in the root partition, edit */etc/fstab* to include the mounting information, and reboot.

Configuring I/O for PFS File Systems

This section tells you how to configure a PFS file system for the best performance, based on the number of I/O nodes available and the typical application usage model. However, much of the performance of a PFS file system depends on tuning the applications that use it. See the *Paragon™ System User's Guide* for information on improving the I/O performance of parallel applications. For general information affecting PFS file systems, refer to "Managing PFS File Systems" on page 10-1.

The Default */etc/pfstab* File

When looking at the default */etc/pfstab*, notice there are nine stripe groups defined. The stripe groups *eight* and *all* are the same. Here is the default */etc/pfstab*. Note that stripe group *one* consists of the directory */home/.sdirs/vol0*. */home* is mounted on partition *rz0f*. Please note that the names *one* through *eight* are only examples. You can use any group names you like.

```
/home/.sdirs/vol0 all one two three four five six seven eight
/home/.sdirs/vol1 all     two three four five six seven eight
/home/.sdirs/vol2 all         three four five six seven eight
/home/.sdirs/vol3 all               four five six seven eight
/home/.sdirs/vol4 all                    five six seven eight
/home/.sdirs/vol5 all                         six seven eight
/home/.sdirs/vol6 all                             seven eight
/home/.sdirs/vol7 all                                   eight
```

Even with one I/O node, it may make sense to have two stripe directories. A stripe directory is in a UFS file system and cannot contain files larger than 2G-1 bytes. For PFS files to be larger than 2G-1 bytes, they must be striped across more than one stripe directory, and each stripe directory must be a separate file system. Then, the portion of a PFS file in a stripe directory is always less than 2G-1 bytes while the entire file may be larger.

Additional I/O Nodes

When you configure the PFS, you may choose to have different I/O nodes used by the PFS and the paging tree. To specify that an I/O node is *not* to be used by the paging tree, add the NOPAGER token to the RAID line in *DEVCONF.TXT*. Then, perform a **reset autocfg**. For example, the following line shows how to identify the I/O with CBS number 00D01 as an I/O node that will not be used for paging.

```
RAID 00D01 ID 0 SW 3.02 LV 3 DC 5 SID 0 RAID3 NOPAGER
```

To configure additional I/O nodes for PFS:

1. Follow the instructions in the previous section under "Configuring an Additional Ethernet Board" on page 11-12 to configure file systems on I/O nodes other than the boot node.

   The partitioning of the RAID subsystem is determined by the disk label. Choose either the non-boot *pfs label or the non-boot *pfspg label.

2. Inclement or add the appropriate entries in */etc/fstab*. For example, if four I/O nodes are added for PFS file striping, the */etc/fstab* entries might be changed to the following:

```
# Local (required to boot mesh) filesystems
#
/dev/io0/rz0a   /                       ufs rw 0 1
/dev/io0/rz0e   /usr                    ufs rw 0 2
#
# Additional local filesystems
#
/dev/io0/rz0f   /home                   ufs rw 0 3
#
# Remote filesystems
/dev/io1/rz0c  /home/.sdirs/vol0       ufs rw 0 5
/dev/io2/rz0c  /home/.sdirs/vol1       ufs rw 0 5
/dev/io3/rz0c  /home/.sdirs/vol2       ufs rw 0 5
/dev/io4/rz0c  /home/.sdirs/vol3       ufs rw 0 5
#
# Parallel filesystems
#
/dev/io0/rz0d  /pfs             pfs rw,stripegroup=four 0 3
#
# NFS filesystems
```

   The difference from the default */etc/fstab* is that the remote file systems are not commented out and the stripe group has been changed to four.

The new configuration has the following consequences:

- The stripe group of the PFS mount has been changed from *one* to *four*. Files created in */pfs* will be striped across stripe group *four*, which is defined in */etc/pfstab* to consist of the directories */home/.sdirs/vol0*, */home/.sdirs/vol1*, */home/.sdirs/vol2*, and */home/.sdirs/vol3*. Because each of the stripe directories is the mount point of a UFS file system on a different I/O node, concurrent striping to four I/O nodes is achieved.

- PFS files are not striped across the boot node, *io0*.

- 2G-1 bytes of each RAID subsystem are used. This is because the stripe directory is mounted on a UFS file system, which has a limit of 2G-1 bytes. This means that maximum use of the data storage capability of the RAID subsystems for PFS use is not attained. However, concurrence is maximized.

If you decide to have two stripe directories per RAID subsystem, the remote mounts in */etc/fstab* might look as follows:

```
#
# Remote filesystems
#
/dev/io1/rz0c /home/.sdirs/vol0 ufs rw 0 5
/dev/io2/rz0c /home/.sdirs/vol1 ufs rw 0 5
/dev/io3/rz0c /home/.sdirs/vol2 ufs rw 0 5
/dev/io4/rz0c /home/.sdirs/vol3 ufs rw 0 5
#
/dev/io1/rz0d /home/.sdirs/vol4 ufs rw 0 5
/dev/io2/rz0d /home/.sdirs/vol5 ufs rw 0 5
/dev/io3/rz0d /home/.sdirs/vol6 ufs rw 0 5
/dev/io4/rz0d /home/.sdirs/vol7 ufs rw 0 5
#
# Parallel filesystems
#
/dev/io0/rz0d /pfs pfs rw,stripegroup=eight 0 3
```

With this configuration, a 128K-byte write will be split between *io1* and *io2*.

The stripe directories must be owned by *root* and have write/execute permissions for *root*, write/execute permissions for *group* and *other*, and have their sticky bits set. The PFS mount point must have read/write/execute permissions for everyone and have its sticky bit set.

The file systems must be mounted before you can change their ownership or permissions. To mount the file systems listed in */etc/fstab*, issue the following command:

```
# mount -a
```

The commands to set the ownership and permissions are as follows (the leading 1 sets the sticky bit):

```
# chown root stripe_directoriues
# chmod 1333 stripe_directoriues
```

The command to do this is as follows:

```
# chmod 1777 mount_point
```

Paging Trees 12

Introduction

This chapter will help you understand and configure *paging trees*, which are used to distribute a Paragon™ system's virtual memory paging load among the system's disk nodes. It includes the following sections:

- Paging tree Concepts.
- Setting up a paging tree automatically.
- Setting up a paging tree manually.
- Implementation of paging trees.
- Increasing default paging file size.

Concepts

A Paragon system is composed of *compute nodes* and *I/O nodes*. Either type of node can also be called a *service node* if it is in the *.service* partition. An I/O node is the same as a compute node except that it has an MIO (Multipurpose I/O) daughtercard which lets it control one or more I/O devices. One special I/O node, called the *boot node*, has a disk drive and an Ethernet interface and is responsible for booting the rest of the system. The boot node also performs several important operating system services while the system is running. Each of the remaining I/O nodes typically has only one I/O device attached to its MIO card, and is referred to as a *disk node*, *Ethernet node*, *HIPPI node*, or *FDDI node* depending on the type of device.

NOTE

A disk node is sometimes referred to as an I/O node. For this discussion, we will refer to I/O nodes as disk nodes.

This chapter is mainly concerned with disk nodes. Disk nodes can have many responsibilities, including:

- Running user processes (if it is in the service or compute partition).
- I/O to UFS file systems.
- I/O to PFS stripe directories (which are also UFS file systems).
- Virtual memory paging for itself (default pager). (Only the boot node can do this.)
- Virtual memory paging for other nodes (vnode pager).

Typically there is more work for disk nodes than there are disk nodes in the system. For example, a single disk node may be responsible for two UFS file systems and a PFS stripe directory and also run user processes. The stability and performance of your system depend on balancing this work among the available disk nodes so that none of the disk nodes is overloaded. Setting up paging trees can help you balance the load.

What Is Paging?

The Paragon operating system includes a feature called *virtual memory*, which lets you use more memory than is physically available on the node. When a program tries to access a memory space that is larger than the node's available free memory, one or more 8K-byte *virtual memory pages* that haven't been referenced recently are *paged out*. This means that their contents are written to disk and replaced with the new data. Later, when the program references data in the paged-out memory section, a different section is paged out and the old data is *paged in* (read back from disk) in its place.

What Is a Pager?

Since paging requires access to a disk drive, each node in the system is assigned a disk node to handle its paging traffic. This disk node is called a *pager*. The relationships of nodes with their pagers are established at boot time and do not change while the system is running. You determine which nodes are pagers and which nodes page to them by editing the *DEVCONF.TXT* or *MAGIC.MASTER* file on the diagnostic station. Each pager stores the pages it receives in a special disk partition called the *paging disk partition* (see "Implementation of Paging Trees" on page 12-15 for more information).

Every node in the system must page to another node (even pagers have pagers). The exception to this rule is the boot node, which is always its own pager. By default, the boot node is also the pager for every other node in the system. In large systems, this paging load can overwhelm the boot node, resulting in slow performance, hangs, or panics. To avoid these problems, you should set up a *paging tree*.

What Is a Paging Tree?

A *paging tree* is a way of configuring some or all of the system's disk nodes as pagers to minimize the impact of paging on any one node. By default, the boot node is the pager for all the other nodes in the system. In a paging tree, you create several pagers and divide up the nodes among them. Because the boot node is the only node that can page to itself, each of the pagers must also have a pager. Typically, all the pagers page to the boot node (this is called a *three-level* paging tree: the compute nodes are the top level, the pagers are the second level, and the boot node is the bottom level).

For example, Figure 12-1 shows a simple paging tree in which two disk nodes (nodes 15 and 23) are pagers for the compute nodes. Half the compute nodes page to each disk node, and the two disk nodes page to the boot node (node 7). Note that in this case the term "compute node" refers to a node in either the compute or service partition.

Setting up a simple paging tree like the one shown in Figure 12-1 can be done automatically, as described under "Setting up a Paging Tree Automatically" on page 12-5.

You can also have all the pagers page to another disk node, which then pages to the boot node (a four-level paging tree); this greatly reduces the paging load on the boot node, but requires an additional pager. To set up a more complicated paging tree like this, see "Setting up a Paging Tree Manually" on page 12-9.

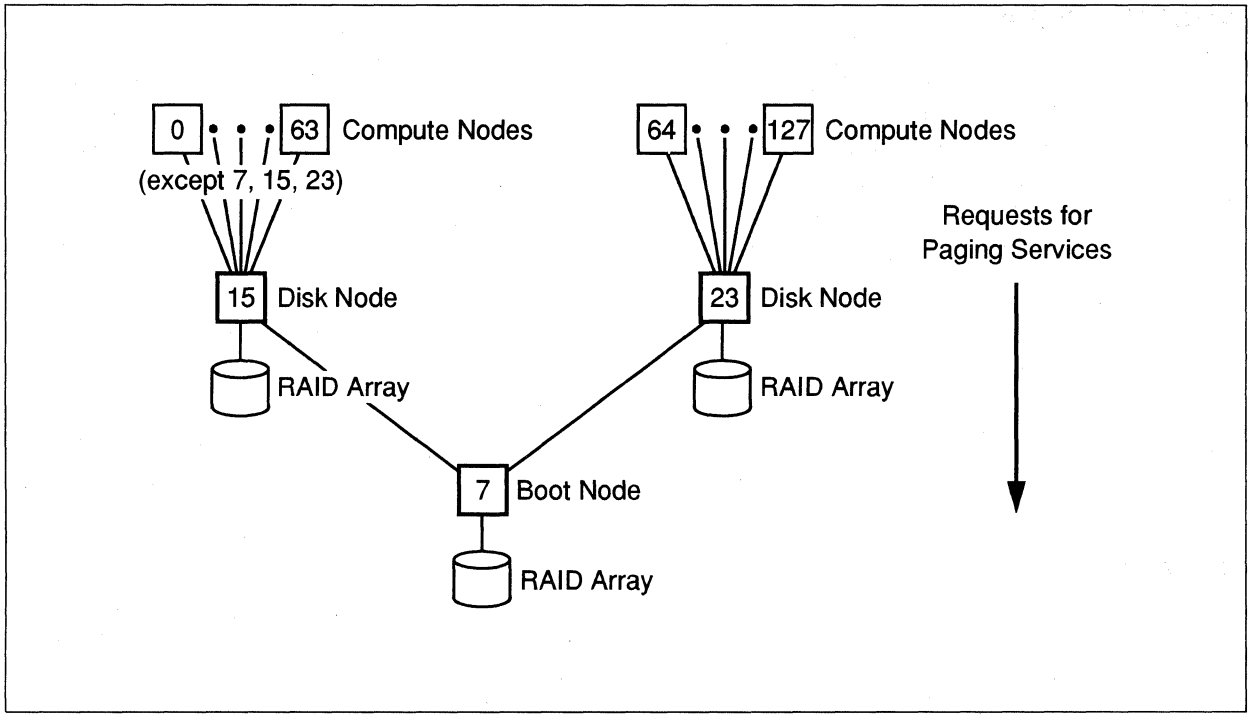


Figure 12-1. Example Paging Tree

Impact of Paging Trees on Disk Nodes

Setting up a paging tree can improve system performance and stability by reducing the paging load on the boot node. However, that load doesn't go away; it is distributed to the pager nodes. This impacts those nodes in one of two ways:

- If a pager node is also used to manage UFS file systems or PFS stripe directories, user disk I/O performance on that node will be reduced whenever paging occurs. You may be able to reduce the impact of this performance degradation on application programs by making sure that the file systems controlled by the pager node are not heavily used. However, if the combined I/O and paging load is too heavy, the pager could hang or panic. (The likelihood of this occurring depends on your system's configuration and application mix.)

- If a pager node is *not* also used to manage UFS file systems or PFS stripe directories, there is no contention between paging and user disk I/O. However, this approach may reduce the number of disk nodes you can use for PFS striping (which reduces the performance and size of your PFS file system). Also, since each RAID array has a capacity of 4G bytes or more and a paging file system can be at most 2G bytes in size, this approach does not fully utilize the RAID array's storage capacity.

You need to keep these considerations in mind when determining how many disk nodes to use as pagers and how to distribute your file systems among the disk nodes. Paging, UFS file systems, and PFS file systems must always be considered together in balancing your system's disk activity.

If a pager also has other responsibilities, such as running user processes in the service partition or controlling an Ethernet interface, these will also increase its load and should be considered when determining how many nodes it can handle as a pager. You should consider putting your I/O nodes into a separate *I/O partition* and setting its permissions so that user processes cannot run on them; this prevents user processes and I/O processes from impacting each other's performance.

Setting up a Paging Tree Automatically

You can have the system set up a simple paging tree like the one shown in Figure 12-1 automatically. This section tells you how.

Characteristics of an Automatically-Generated Paging Tree

By default, an automatically-generated paging tree has the following characteristics:

- All disk nodes listed in the *SYSCONFIG.TXT* file on the diagnostic station are used as pagers.
- The device *rz0b* is used as the paging disk partition on each pager.
- All pagers page to the boot node.
- All remaining nodes are evenly divided among the pagers.

Customizing an Automatically-Generated Paging Tree

You can customize the automatically-generated paging tree in two ways:

- To prevent a disk node from being used as a pager, add the token **NOPAGER** to the node's **RAID** line in *DEVCONF.TXT*, then create a new *SYSCONFIG.TXT* with the command **reset autocfg**.
- To specify a paging disk partition other than *rz0b* for a pager, add the string **PAGE_TO** *device* (for example, `PAGE_TO rz1a`) to the pager's **RAID** line in *DEVCONF.TXT*, then create a new *SYSCONFIG.TXT* with the command **reset autocfg**. If you do this, be sure the specified device (for example, */dev/io3/rz1a*) exists.

To customize your paging tree still further, see "Setting up a Paging Tree Manually" on page 12-9.

Procedure for Setting up a Paging Tree Automatically

Without a paging tree, all of the compute nodes in your system page to the boot node. This is the default and is called a two-level paging tree. Using the default for large systems may result in poor performance.

If your Paragon system has I/O nodes in addition to the boot node, you can use a three-level paging tree to designate these additional I/O nodes as paging nodes. Paging trees up to four levels are supported.

Figure 12-1 shows a three-level paging tree in which two I/O paging nodes (15 and 23) accept paging from the compute nodes. Nodes 15 and 23 in turn page to the boot node (7). It is called three-level because the first level consists of the compute nodes paging to the I/O nodes; the second level is the I/O nodes paging to the boot node; the third level is the boot node paging to the RAID array.

A three-level paging tree (as shown in Figure 12-1) where all the non-boot I/O nodes page to the boot node can be obtained by specifying the **-P1** option for **bootpp** in the **reset** script in */usr/paragon/boot* on the diagnostic station. The default paging partition is *rz0b*.

1. Log into the diagnostic station as *root*. Ensure that the partitions *rz0b* on your RAID subsystems do not have file systems mounted on them. Note that, unless you change the default paging partition, all data on *rz0b* will be lost.

2. Ensure that the device nodes for all the RAID subsystems are created. For example, if you have three I/O nodes, the following device nodes must exist on the Paragon system in */dev*: *io0*, *io1*, and *io2*.

3. The **reset** script in */usr/paragon/boot* contains the following line:

   ```
   $BOOTPP -W -Z\
   ```

   Edit the line to be

   ```
   $BOOTPP -P1 -W -Z\
   ```

   Do not issue **bootpp** from the command line. The proper procedure is to edit the **reset** script and reset your system. The paging tree is created when you reset your system.

   This method defines the default paging partition on the I/O nodes as *rz0b*. To have an I/O node use a paging partition other than the default, perform the following steps.

   A. Edit *DEVCONF.TXT* and put the PAGE_TO token on the RAID line identifying the I/O node that will do the paging. If PAGE_TO is left out the default *rz0b* is used. For example, the following line shows how to identify the I/O with CBS number 00A01 as a paging node that uses the *rz0c* partition for paging.

      ```
      RAID 00A01 ID 0 SW 3.04 LV 3 DC 5 SID 0  RAID3 PAGE_TO rz0c
      ```

B. Then issue a **reset autocfg**. This creates a new *SYSCONFIG.TXT*. It's best to put the PAGE_TO token in *DEVCONF.TXT* because if it's not there, then whenever you do a **reset autocfg**, you will lose the paging information.

C. Another way (which is not the recommended way) is to set the bootmagic string PAGE_TO in */usr/paragon/boot/MAGIC.MASTER*. If you then perform a **reset**, the specification in *MAGIC.MASTER* overrides that in *DEVCONF.TXT* or *SYSCONFIG.TXT*. The format of this variable is

PAGE_TO=<*node_nbr*, *node_nbr*, . . .>*part*:<*node_nbr*, *node_nbr*, . . .>*part*. .

For example, if you wanted the paging tree shown in the diagram for this example to page to partition *rz0c* instead of *rz0b*, add the following line to */usr/paragon/boot/MAGIC.MASTER*:

```
PAGE_TO=<15>rz0c:<23>rz0c
```

NOTE

In the *SYSCONFIG.TXT*, *MAGIC.MASTER*, or *DEVCONF.TXT* files, the PAGE_TO specification has no effect on the boot node. The boot node always uses the default *rz0b* as the paging partition.

4. Connect to the Paragon system with the **console** command.

```
DS# cd /usr/paragon/boot
DS# ./console
          .
          .
          .
login:
```

5. Login as *root* to the Paragon system and shut down the Paragon system. Then, return to the diagnostic station prompt by typing **~.** (or **~~.** if you are logged in remotely to the diagnostic station). The key sequence **~q** also works and does not require that you keep track of the number of remote logins. Then, use the **reset** script to reboot the system.

When the **reset** command has completed and the prompt returns, enter multiuser mode by pressing **<Ctrl-D>**. Note that the **reset** script automatically enters multiuser mode if in the *MAGIC.MASTER* file contains RB_MULTIUSER=1 .

```
# cd /
# shutdown now
# halt
# ~.
DS# ./reset
          .
          .
          .
# <Ctrl-D>
```

Increasing Default Paging File Size

By default, the **install** script creates a 64M-byte page file size for the default pager on the boot node. If you want to increase the size of the paging file after installation, follow the instructions in this section.

If the system produces an error similar to `Paging File Exhausted`, it could indicate the paging size is too small. The following procedure shows how to increase the page size which, in this example, is increased to 512M bytes. The new paging file is placed in the */home* partition because the */root* partition is not large enough for such a file.

NOTE

Increasing the page size to 512M bytes can take up to thirty minutes. A smaller page size will take less time. Before you increase page size, you should make sure there is enough room in the home partition.

1. Log into the Paragon system as *root*. Create the paging file in the home partition.

```
# /sbin/create_pf 512M /home/paging_file
```

2. You also need to make a symbolic link from the old paging file in the root partition to the new paging file in the home partition. But the microkernel has difficulty following a symbolic link to the home partition */dev/io0/rz0f*. So you need to make a hard link from */dev/io0/rz0f* to */dev/rz0f*.

```
# ln /dev/io0/rz0f /dev/rz0f
```

3. Shutdown the Paragon system.

```
# cd /
# halt
# ~q
```

4. The shutdown returns you to the diagnostic station prompt. Reset to single-user mode. Note that the **reset** script automatically enters multiuser mode if the *MAGIC.MASTER* file contains RB_MULTIUSER=1 .

```
DS# ./reset
```

5. Now, on the Paragon system, make the root partition writable, save the original version of the paging file, and then make a symbolic link between the new paging file and the old one.

```
# mount -u /
# mv /mach_servers/paging_file /mach_servers/paging_file.orig
# ln -s /dev/rz0f/paging_file /mach_servers/paging_file
```

6. Reboot the Paragon system to multiuser. If there is any problem with the new paging file, the system may complain not being able to find the paging file and it prompts for the paging file's name. If that happens, enter */mach_servers/paging_file.orig* to boot the system. Once the system has no problem booting with the new paging file, you may remove */mach_servers/paging_file.orig*.

Setting up a Paging Tree Manually

The automatically-generated paging tree produced by adding the **-P1** switch to the **reset** script has certain limitations. It is always a three-level paging tree, and always divides the compute nodes evenly among the pagers.

Figure 12-2 shows an example of a more complicated paging tree. This type of paging tree cannot be configured automatically in the current release; this section tells you how to set up a paging tree like this by hand.

In Figure 12-2, two disk nodes (nodes 47 and 79) are pagers for the compute nodes and one disk node (node 63) is the pager for the service nodes (nodes 95, 111, and 127). These three pagers page to a fourth disk node (node 31), and only node 31 pages to the boot node (node 15). This paging tree dramatically reduces the paging load on the boot node, because the boot node is only a pager for one other node. (Note that the load is handled by node 31 instead; that node might have so much paging to do that it would be unable to do any other work.) This example assigns more than 32 nodes to some pagers; this is not advised, but is done here to simplify the example. In a real system of this size, you would probably have more disk nodes to use as pagers.

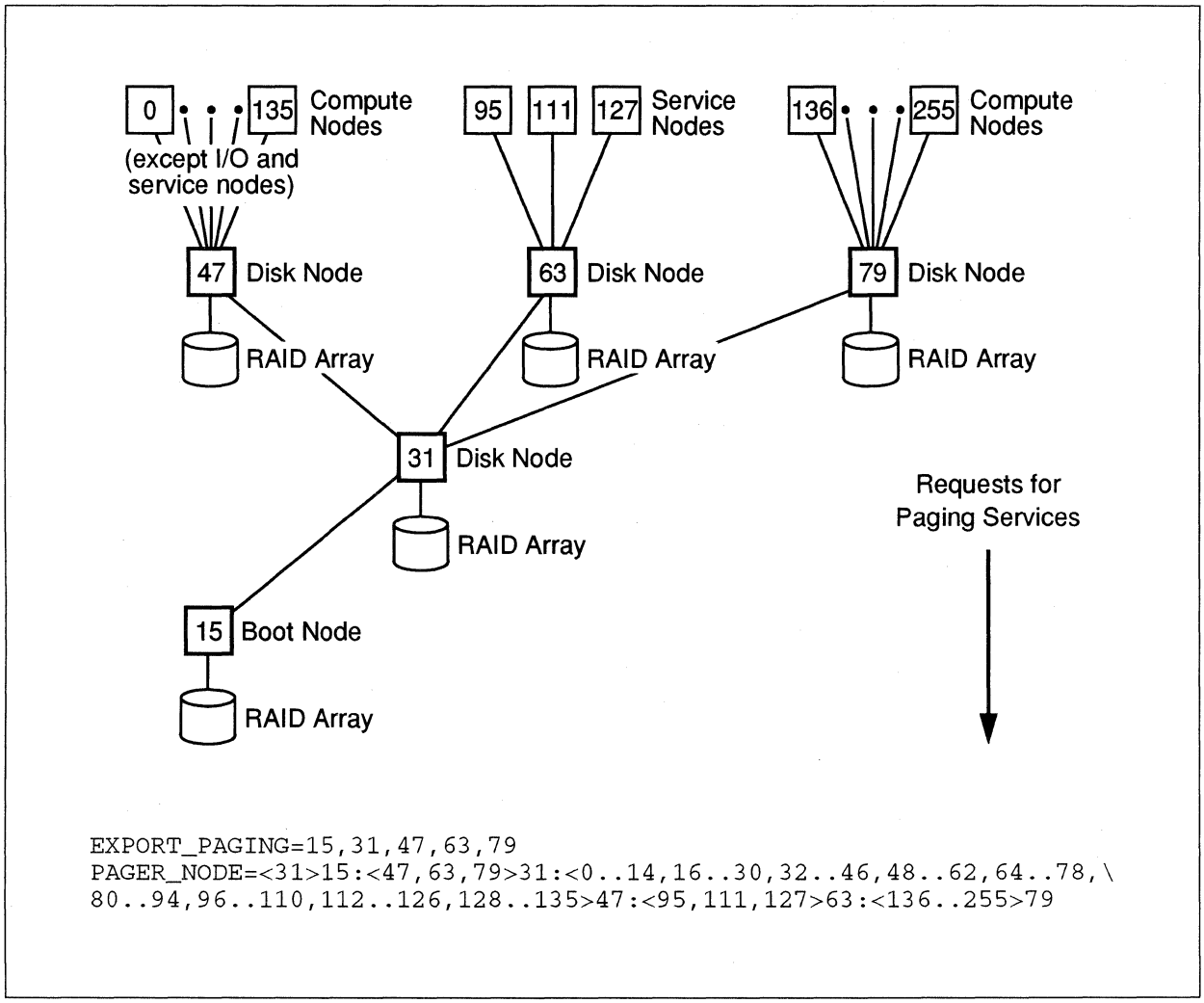


Figure 12-2. More Complicated Example Paging Tree

This example paging tree also divides the non-pager nodes unevenly among the pagers; the service nodes page to one pager, and the compute nodes are divided among the remaining pagers. It may seem unbalanced to have two pagers handling many compute nodes while one pager handles only a few service nodes, but it is more balanced than it looks because service nodes generally page more than compute nodes. However, this is only an example; in a real system you might have a mix of compute and service nodes assigned to each pager in order to achieve the best possible balance.

To configure a paging tree like this, you must edit the *MAGIC.MASTER* file, as discussed under "Procedure for Setting Up a Paging Tree Manually" on page 12-13. The *EXPORT_PAGING* and *PAGER_NODE* lines from the *MAGIC.MASTER* file that implement this paging tree are shown at the bottom of the figure (note the use of the backslash continuation character to split the *PAGER_NODE* line over two lines of the file).

Guidelines for Setting Up a Paging Tree Manually

There is no simple answer to the question "how do I set up the best paging tree for my system?" The answer depends on the number of nodes of different types in your system, the amount of physical RAM on each node, the distribution of UFS and PFS file systems among the disk nodes, and the way your users' applications use memory and perform disk I/O. You have to use your knowledge of your application mix to balance the load. Your goal is to divide up your system's total disk I/O load (both user disk I/O and paging) among the disk nodes so that no node is either idle or overloaded under normal conditions.

General Guidelines

- Try not to assign more than 32 nodes to a single pager. If the nodes page heavily, you should assign fewer than that. If this is not possible, try to balance the load as much as possible.

- Compute nodes may or may not perform a lot of paging, depending on your application mix. For example, a single application can cause every node it runs on to page if it allocates more than the available free memory, but with a smaller application the compute nodes would not page at all. On the other hand, I/O nodes and service nodes generally page a lot, because they provide multiple services to other nodes which use a lot of memory. This means that you may be able to assign many compute nodes to a single pager, but should assign only a few I/O or service nodes to each pager.

- If the number of pagers is large (more than 8), or if you have set up a three-level paging tree and the boot node still appears to be overloaded, try setting up a four-level paging tree (where the pagers all page to another pager, which pages to the boot node). If you do this, you should try to avoid giving that second-level pager any additional I/O work to do.

- Try not to mix paging and PFS. If nodes that manage PFS stripe directories are also pagers, PFS performance will suffer whenever paging occurs to those nodes. However, since each RAID array has a capacity of 4G bytes or more and a paging file system can be at most 2G bytes in size, nodes that perform only paging do not fully utilize the RAID array's storage capacity.

Examining Your System's Paging Activity with SPV

To help you determine your system's memory and disk activity, you can use **spv**, the System Performance Visualization tool (see the *Paragon™ System Performance Visualization Tool User's Guide* for more information on this tool). When you select "Zoom to Nodes" from the "Zoom" menu, you see the display shown in Figure 12-3 for each node. The figure identifies the parts of this display that are most important for configuring paging trees. (Note that the I/O block in the upper right appears only for an I/O node.)

SPV can also be configured to show the memory usage, wired memory, or page faults per second for each node (instead of the CPU utilization) in the CPUs, Mesh, and Values displays. This allows viewing the memory usage of the whole system at once.

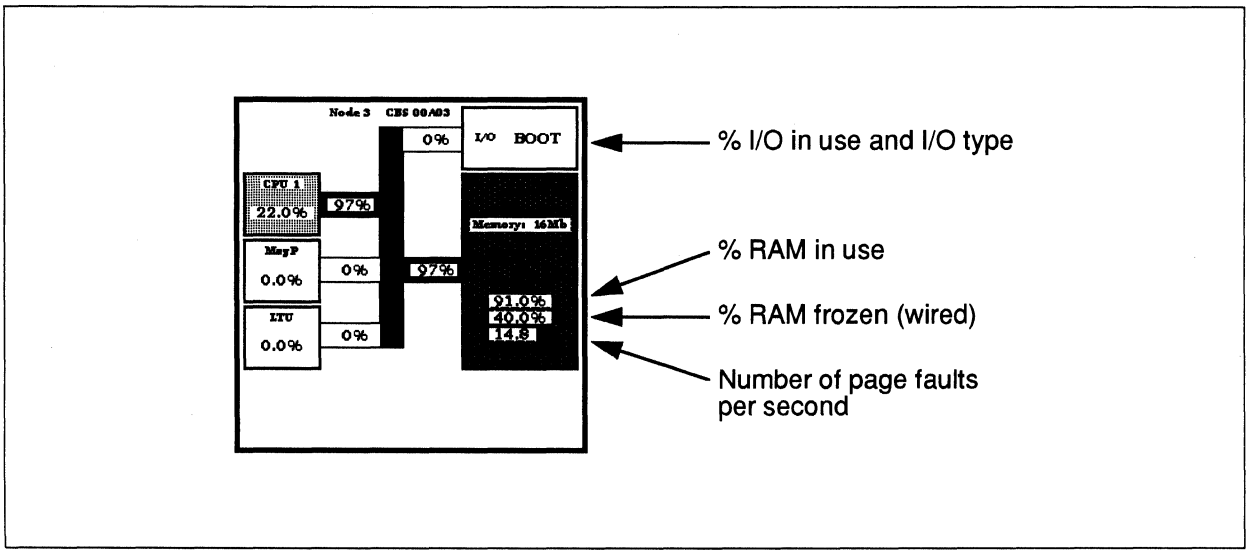


Figure 12-3. SPV Node Display

You should examine your system under a variety of different application loads. (SPV can record system activity as it occurs and then play it back later at your convenience; see the *Paragon™ System Performance Visualization Tool User's Guide* for information on how to do this.) As you examine the system, keep in mind which nodes are currently pagers and which nodes page to them. It might be helpful to have a copy of your current *MAGIC.MASTER* file nearby.

- When examining a pager, note the percentage of RAM and I/O in use. A pager that is heavily loaded will show a high percentage of both. If the percentage of RAM in use is consistently 90% or more, the pager may be overloaded. An overloaded pager will also show a large number of page faults per second (several hundred or more).

- When examining a non-pager, note the number of page faults per second. The more page faults, the heavier the load that node is placing on its pager. (This is a generality; not every page fault actually results in a page being sent to or requested from the pager.)

The paging load on a pager is determined by the total number of page faults per second of all the nodes that page to it. Use the data presented by SPV to help you balance the load (paging and I/O) by reassigning nodes to different pagers. Keep shifting nodes around until the pagers are approximately equally loaded under most typical usage scenarios.

You can also use the command **vm_stat -n** *node* to examine the virtual memory statistics for a single node in more detail. If you don't use the **-n** switch, **vm_stat** gives statistics for the node it runs on, which is usually a node in the service partition. See **vm_stat** in the *Paragon™ System Commands Reference Manual* for more information.

SUNMOS Considerations

The SUNMOS operating system does not support virtual memory, so nodes running SUNMOS should not appear anywhere in the paging tree. However, if a node that was running SUNMOS is later switched to operating system, you must be sure to restore that node to the paging tree at the same time. You may have to re-balance the paging tree when you do this.

Procedure for Setting Up a Paging Tree Manually

If you want to configure a four-level paging tree like the one shown in Figure 12-2 on page 12-10, or if you want to assign more compute nodes to some pagers and fewer compute nodes to other pagers, you have to set up your paging tree by hand. To do this, you must edit the *MAGIC.MASTER* file on the diagnostic station.

NOTE

Settings in the *MAGIC.MASTER* file override all **NOPAGER** and **PAGE_TO** directives in the *SYSCONFIG.TXT* file.

If you set up a paging tree as described in this section, any automatically-generated paging tree is ignored. This means that if you add or remove disk nodes or otherwise change your system's configuration, you must change the definition of your paging tree in the *MAGIC.MASTER* file to match the new configuration.

The following procedure shows you how to set up a paging tree by hand. It uses the paging tree shown in Figure 12-2 on page 12-10 as an example. The example uses a two-cabinet system with the boot node at node number 15 and additional disk nodes at nodes 31, 47, 63, and 79.

NOTE

This example uses the (nonexistent) **x** partition on each disk drive as the paging disk partition.

You should examine each of your disk drives with **disklabel** and select the appropriate paging disk partitions for your system. See "Paging Disk Partitions" on page 12-17 for more information.

1. Log into the diagnostic station and the Paragon system as *root* (preferably in two separate windows).

2. On the Paragon system, make sure that the device special files for each paging disk partition have been built. For example, the following command shows which I/O nodes have the device special file for partition x:

```
# ls -l /dev/io*/rz0x
brw-r--r--   1 root     system     15:  3,  1 Jan 04 13:04 /dev/io0/rz0x
brw-r--r--   1 root     system     31:  3,  1 Jan 04 13:04 /dev/io1/rz0x
brw-r--r--   1 root     system     47:  3,  1 Jan 04 13:04 /dev/io2/rz0x
brw-r--r--   1 root     system     63:  3,  1 Jan 04 13:04 /dev/io3/rz0x
brw-r--r--   1 root     system     79:  3,  1 Jan 04 13:04 /dev/io4/rz0x
```

The node number for each device is shown followed by a colon.

3. On the diagnostic station, check the file */usr/paragon/boot/bootmagic* to make sure the *BOOT_DISK_NODE_LIST* line contains all disk nodes, including the boot node. In this example, the *BOOT_DISK_NODE_LIST* line looks like this:

```
BOOT_DISK_NODE_LIST=15,31,47,63,79
```

Note that this line is generated automatically from *SYSCONFIG.TXT*, and should *not* be present in the *MAGIC.MASTER* file.

4. On the diagnostic station, edit the *EXPORT_PAGING*, *PAGER_NODE*, and *PAGE_TO* lines of the file */usr/paragon/boot/MAGIC.MASTER* to configure your paging tree:

   A. Modify the *EXPORT_PAGING* line so that it includes the node numbers of the boot node and your new pager nodes. This line has the following format:

   ```
   EXPORT_PAGING=nodelist
   ```

   B. Modify the *PAGER_NODE* line so that the new pager nodes page to the boot node, and the rest of the nodes page to the new pager nodes. This line has the following format:

   ```
   PAGER_NODE=<nodelist>pager[:<nodelist>pager]...
   ```

   C. Modify the *PAGE_TO* line to specify the paging disk partitions for each pager node.

   The *PAGE_TO* line is optional. If you omit it, the **PAGE_TO** tokens in *SYSCONFIG.TXT* (generated from *DEVCONF.TXT* by **reset autocfg**) determine the paging disk partition for each node. If the paging disk partition is not specified for a node, the default is *rz0b*.

No spaces may appear in the *EXPORT_PAGING*, *PAGER_NODE*, or *PAGE_TO* line. Each *nodelist* is a single node number, a range of nodes in the form *number..number*, or a comma-separated list of node numbers and/or ranges. Each *pager* is the node number of the pager node for the nodes in the preceding *nodelist*, and each *partition* is the basename of the device special file of the paging disk partition for the nodes in the preceding *nodelist*.

For example, the following *EXPORT_PAGING*, *PAGER_NODE*, and *PAGE_TO* lines implement the paging tree shown in Figure 12-2 on page 12-10:

```
EXPORT_PAGING=15,31,47,63,79
PAGER_NODE=<31>15:<47,63,79>31:<0..14,16..30,32..46,\
48..62,64..78,80..94,96..110,112..126,128..135>47:\
<95,111,127>63:<136..255>79
PAGE_TO=<15,31,47,63,79>rz0x
```

Note the use of the backslash continuation character to split the *PAGER_NODE* line over three lines of the file.

NOTE

In the current release, paging disk partitions do *not* have to appear in the */etc/fstab* file. Instead, they are listed in the *PAGE_TO* bootmagic string.

5. On the Paragon system, shut the system down:

```
# cd /
# shutdown now
# halt
```

6. On the diagnostic station, reboot the Paragon system:

```
DS# ./reset
```

Your new paging tree is now installed. Try running some large applications while observing the system with **spv** to see whether the load is better balanced than it was before. You may have to try several paging tree configurations before you find the optimum arrangement for system performance and stability.

Implementation of Paging Trees

This section discusses how paging trees are implemented in operating system. If you're not interested in the technical details, you can skip to the next chapter. This section assumes that you understand a few basic operating system operating system concepts such as *server* and *kernel*.

Default Pagers and Vnode Pagers

Each node has an entity called the *default pager*: the part of the kernel that handles paging requests from the node itself. When a node needs to page some of its memory out, it puts the contents of the page in its default pager for storage; when the node needs to page some memory in, it gets the contents of the page from its default pager.

On the boot node, the default pager works by reading and writing the file */mach_servers/paging_file* on the node's own disk. On every other node, the default pager works by sending Mach IPC messages to a *vnode pager* on another node. The vnode pager is part of the server. It exists only on pager nodes; non-pager nodes do not have a vnode pager.

A vnode pager handles paging requests from *other nodes* (never the node on which it runs). When a vnode pager receives a page from another node, it stores the page in the paging disk partition of the vnode pager's disk. When it receives a request for a page, it retrieves the page from its disk and sends the contents of the page back over the mesh to the requesting node.

Paging trees are implemented by connecting non-pager nodes' default pagers to vnode pagers on pager nodes; the pager nodes' default pagers are connected to the vnode pager on the boot node. Only the boot node actually pages to its own disk. The relationship between the various types of pagers is shown in Figure 12-4.

As you can see from Figure 12-4, a paging request never has to travel to more than one other node to be fulfilled. However, heavy paging demand on a pager node can also cause paging demand on that node's pager (typically the boot node). This occurs because paging can consume memory on the pager node.

If a page, or a request for a page, arrives at a pager node while the vnode pager is busy, the incoming page or request must be temporarily buffered in memory. If many nodes assigned to the same pager all page at once, hundreds of pages/requests might get backed up in this way. Each backed-up request takes up RAM on the pager node. When so many requests are backed up that the node's memory is full, the pager node begins paging itself. At this point the node may be in trouble.

While the vnode pager continues trying to handle incoming pages and requests, at the same time the kernel is sending pages from the node's memory to its default pager in order to make room for more incoming pages. This steals CPU cycles from the vnode pager, which makes it run slower and increases the backlog. If the pager in question is the boot node, this also causes contention for the disk, which makes the vnode pager run still slower. If the load continues and the two pagers can't free up pages faster than new pages are coming in, the node eventually hangs or panics.

This shows how a well-balanced paging tree can increase system stability as well as performance. If the non-pager nodes are carefully distributed among the pagers so that no pager ever receives more requests than it can handle, the system will stay up longer.

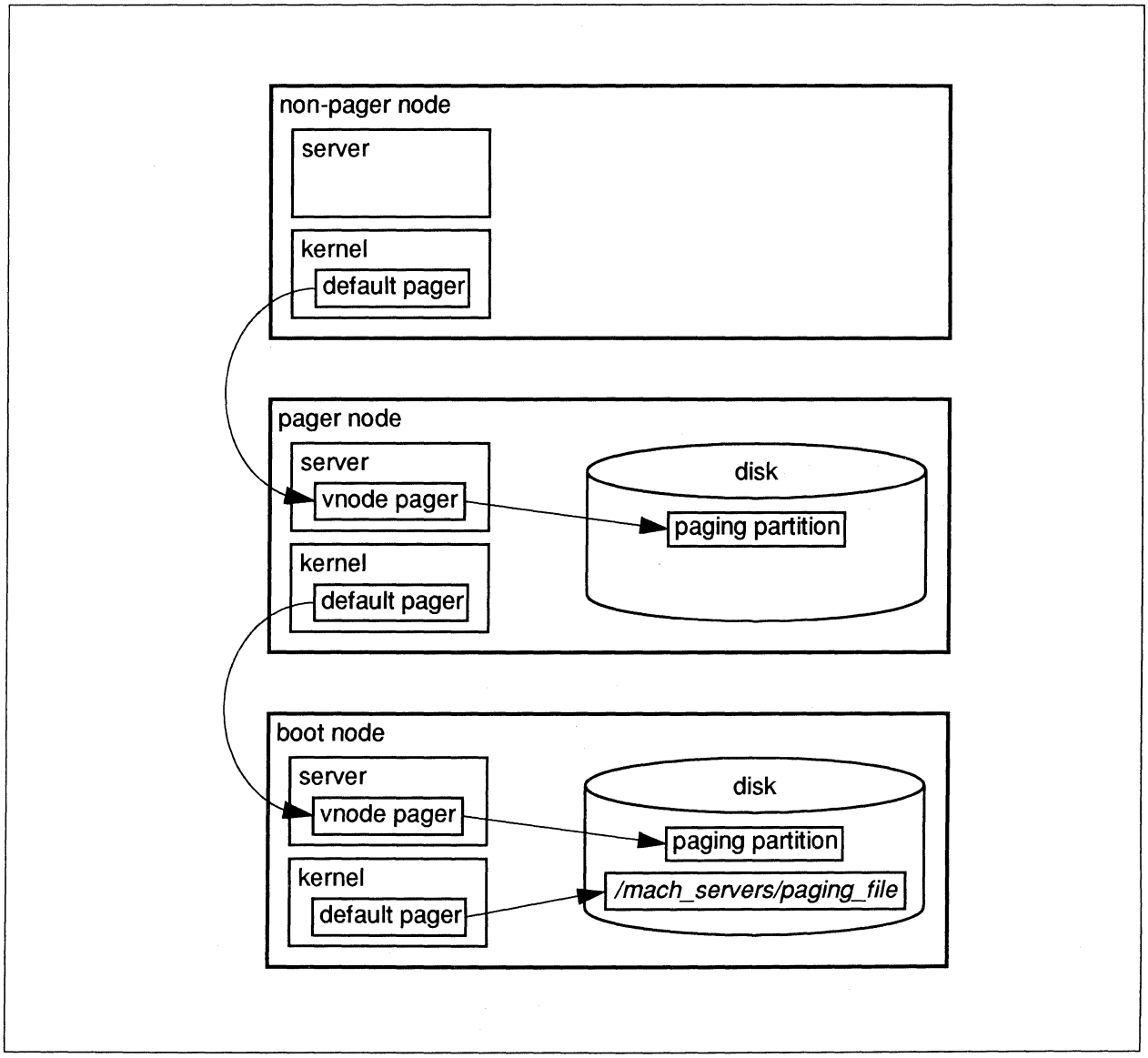


Figure 12-4. Implementation of Paging Trees

Paging Disk Partitions

Each pager node stores the pages it receives in a special disk partition called the *paging disk partition* on its disk. A paging disk partition should have an 8K disk block size and should be at least as large as the combined physical memory of all the nodes that are assigned to the pager. For example, if the pager supports 32 nodes with 32M bytes of RAM each, the paging disk partition should be at least (32 * 32M) bytes, or 1G byte. A paging disk partition cannot be used for anything else.

You can use the **disklabel** command to examine the partitions of a disk and determine which ones are appropriate. The device you specify to **disklabel** can be any device associated with the disk. For example, to see the partitions of the disk with SCSI ID 0 on I/O node 0 (the boot node), you can use the following command:

```
# disklabel -r /dev/io0/rrz0a
```

CAUTION

Examine all disks carefully before deciding which partitions to use for paging. Not all disks may be partitioned in the same way.

For example, some of the examples of paging trees in the Paragon system documentation show partition **b** as a paging disk partition. However, some of the examples of configuring PFS file systems show partition **b** as a PFS stripe file system. Your system may have some disks configured for paging, and others configured for PFS. If you accidentally use a disk partition that is currently in use for PFS or UFS data for paging, the contents of that partition will be destroyed.

Increasing Default Paging File Size

If the system produces an error similar to `Paging File Exhausted`, it could indicate the size of the paging file used by the boot node's default pager is too small. The following procedure shows how to increase the page size which, in this example, is increased to 512M bytes. The new paging file is placed in the */home* partition because the / (root) partition is not large enough for such a file.

1. Log in or **rlogin** to the diagnostic station as *root*.

2. Connect to the Paragon system with the **console** command. The **console** command is a script that the **reset** script created in */usr/paragon/boot* the last time **reset** was run. **console** uses whatever the console connection was at the time and supports the **async**, **fscan**, and the **scanio** console interfaces. You may have to press a carriage return after issuing **console** to get a prompt.

```
DS# cd /usr/paragon/boot
DS# ./console
          •
          •
          •
login:
```

3. Log in as *root* to the Paragon system, unmount all file systems, and use the **halt** command to shut down the Paragon system. Then, return to the diagnostic station prompt by typing **~q**. (The key sequence **~.** also works, but if you are remotely logged into the diagnostic station you must enter **~~.** instead to prevent terminating the **rlogin** session.) Then, issue a **reset ramdisk**, followed by the following series of commands:

```
DS# ./reset ramdisk
<ramdisk> mount -u /
<ramdisk> fsck -y /dev/rrz0a
<ramdisk> mount -w /dev/rz0a /root
<ramdisk> fsck -y /dev/rrz0f
<ramdisk> mount -w /dev/rz0f /home
<ramdisk> cd /home
<ramdisk> /root/sbin/create_pf 512M paging_file
<ramdisk> chmod 600 paging_file
<ramdisk> cd /root/dev
<ramdisk> ln io0/rz0f rz0f
<ramdisk> cd /root/mach_servers
<ramdisk> mv paging_file paging_file.orig
<ramdisk> /root/sbin/ln -s /dev/rz0f/paging_file paging_file
<ramdisk> cd /
<ramdisk> sync
<ramdisk> umount /root
<ramdisk> umount /home
<ramdisk> ~q
DS# ./reset
```

When the **reset** command has completed and the prompt returns, enter multiuser mode by pressing **<Ctrl-D>**. Note that the **reset** script automatically enters multiuser mode if RB_MULTIUSER=1 in the *MAGIC.MASTER* file.

NOTE

Increasing the page size to 512M bytes can take up to thirty minutes. A smaller page size will take less time. Before you increase page size, you should make sure there is enough room in the root partition.

Error Logging 13

This chapter describes the basic administrative tasks for configuring and starting system error logging.

Overview

The **syslogd** daemon, the error logging daemon, is a general message collector and distributor for status information and single-line error messages produced by various system programs. The **syslogd** daemon frees individual programs from having to handle message logging themselves.

The **syslogd** daemon routes the messages it receives according to the information specified in the */etc/syslog.conf* configuration file. This file is created and maintained by the system administrator. The information in *syslog.conf* can specify that messages be directed as follows:

- Written to all users.
- Written to selected users.
- Appended to a file.
- Forwarded to one or more other systems.

Message priority is determined by specifying both of the following:

- The part of the system that generates the message (facility).
- The severity of the message (level).

The **syslogd** daemon logs all messages that correspond to the specified facility and level, plus all messages of greater severity for that facility. Levels of severity are described in "Selectors" on page 13-2.

Creating and Modifying the Error Logging Configuration File

The *syslog.conf* file contains the specifications for the files, users, and systems to which **syslogd** sends messages. Each line of the *syslog.conf* file contains a selector that specifies a set of messages by facility and level, and a destination that specifies how **syslog** should handle these messages.

The destination field is separated from the selector field by one or more tabs (do not use blanks). The **syslogd** daemon ignores lines that are blank and lines that begin with a number sign (#).

Selectors

Selectors consist of lists of priorities separated by semicolons (;) in the following form:

facility[,*facility* ...].*level*[;*facility.level* ...]

You can specify more than one facility by using commas as separators. You can specify all facilities with an asterisk (*).

Valid facilities are as follows:

kern Messages generated by the kernel. These messages cannot be generated by any user process.

user Messages generated by user processes. This is the default facility.

mail Mail system.

daemon System daemons.

auth The authorization system (**login**, **su**, **getty**, and so on).

lpr The line printer spooling system (**lpr**, **lpd**, and so on).

local0 Reserved for local use, as is **local1** through **local7**.

mark Receives a message of priority information every 20 minutes, unless a different interval is specified for **syslogd** with the **-m** flag.

Valid levels, from highest to lowest severity, are as follows:

emerg or **panic** A panic condition. This is normally broadcast to all users.

alert A condition that should be immediately corrected, such as a corrupted system database.

crit A critical condition, such as a hard device error.

error or **err** Errors.

warning or **warn**
Warning messages.

notice Conditions that are not error conditions, but should possibly be handled as special cases.

info Informational messages.

debug Messages that contain information normally of use only when debugging a program.

You can use the level **none** to disable a specific facility. For example, ***.debug;mail.none** prevents **syslogd** from sending level **debug** messages generated from the mail facility. All other facilities receive level **debug** messages.

Destinations

The destination part of each line in the *syslog.conf* configuration file specifies where messages should be logged. The four available forms are as follows:

- A full pathname (beginning with a leading slash [/]). **syslogd** opens the file in **append** mode. (For additional information, see "Creating a Daily Directory" on page 13-5.)
- A hostname preceded by an at sign (@). **syslogd** forwards messages to **syslogd** on the named host.
- A comma-separated list of users. **syslogd** writes messages to the specified users if they are logged in.
- An asterisk (*). **syslogd** writes messages to all users who are logged in.

For example, a *syslog.conf* file might contain the following two lines:

```
*.notice;mail.info      /var/spool/adm/syslog
kern.err                @venus
```

The first line in the preceding example makes **syslogd** log all messages of level **notice** or higher and all mail messages of level **info** or higher (that is, all messages except for those of level **debug**) in the */var/spool/adm/syslog file*. The second line makes **syslogd** forward all kernel messages of level **err** or higher to the system *venus*.

The Default Configuration

If no configuration file is present, **syslogd** takes a default configuration. The built-in default values are as follows:

```
*.err                /dev/console
*.panic                 *
```

These built-in values mean that all messages of level **err** or higher are logged on the system console and all messages of level **panic**, which come from the kernel, are sent to all logged-in users. The **syslogd** daemon does not write any information to files.

It is not recommended that error logging be done in the preceding way. Instead, you should create a *syslog.conf* file and keep log files of the messages that **syslogd** retrieves. This is described in the next section.

Activating the Error Logging Daemon

The error logging daemon **syslogd** is generally activated automatically by the following command line towards the end of the */etc/rc* start-up file:

```
/usr/sbin/syslogd
```

To specify an alternate configuration file, use the **-f** flag as follows:

```
/usr/sbin/syslogd -fconfig_file
```

NOTE

You must also have mounted the */var/adm* directory; otherwise, the **syslogd** daemon does not work as expected.

The **syslogd** daemon reads messages from the domain socket */dev/log* (messages from other programs using the **openlog**, **syslog**, and **closelog** calls), from an Internet domain socket specified in */etc/services*, and from the special device */dev/klog* (kernel messages only). Before you start up **syslogd**, you should be sure that the */dev/klog* socket exists. If */dev/klog* does not exist, you can create it with the following command line:

```
mknod /dev/klog c major minor
```

The *major* and *minor* arguments are decimal or octal numbers that specify the device type (*major*) and unit, drive, or line number (*minor*).

If */dev/klog* does not exist, **syslogd** cannot log kernel messages. The */dev/log* socket is automatically created by **syslogd**.

Creating a Daily Directory

As mentioned earlier, destinations for logged messages can be specified with full pathnames (beginning with a leading slash [/]). The **syslogd** daemon then opens the specified file(s) in **append** mode. If the pathname to a **syslogd** log file specified in *syslog.conf* is */var/adm/syslog.dated/file*, **syslogd** inserts a date directory, thereby producing a day-by-day account of the messages received directly above *file* in the directory structure. Typically, you will want to separate and divert messages into files such as *kern.log*, *mail.log*, *lpr.log*, and *debug.log*, according to facility.

For example, if you want a daily mail log in a file called *mail.log*, specify */var/adm/syslog.dated/mail.log* as the log file pathname in *syslog.conf* opposite the selector for **mail** messages. The **syslogd** daemon then logs these messages into daily directories such as */var/adm/syslog.dated/06-Aug-12:10/mail.log*, */var/adm/syslog.dated/07-Aug-02:32/mail.log*, and so on.

If some pathname other than */var/adm/syslog.dated/file* is specified as the pathname to the logfile, **syslogd** does not create the daily date directory. For example, if you specify */var/adm/syslog/mail.log* (without the **.dated** suffix after **syslog**), **syslogd** simply logs messages to the *mail.log* file and allows this file to grow indefinitely.

Examining Error Logging Files

The **syslogd** daemon acts as a central routing facility for messages whose formats are determined by the programs that produce them because the message format for error messages and status information is not standardized.

A few sample log files follow:

A sample *kern.log* file:

```
Aug 8 17:21:50 tempest vmunix: NFS server petunia:/latest not responding
Aug 8 17:21:50 tempest vmunix: NFS server petunia:/latest not responding,
retrying
Aug 8 17:21:50 tempest vmunix: NFS server petunia:/latest not responding,
retrying
Aug 8 17:22:46 tempest vmunix: NFS server petunia:/latest OK
Aug 8 17:22:46 tempest vmunix: NFS server petunia:/latest responded
Aug 22 12:28:48 tempest vmunix: NFS server android:/usr/var/spool/mail not
responding, retrying
Aug 22 12:30:22 tempest last message repeated 3 times
Aug 22 12:31:24 tempest vmunix: NFS server android:/usr/var/spool/mail OK
Aug 22 12:31:25 tempest vmunix: NFS server android:/usr/var/spool/mail responded
Sep 5 11:20:17 tempest vmunix: /users: file system full
Sep 5 12:11:38 tempest vmunix: Ethernet: late collision, check transceiver
Sep 5 13:09:44 tempest vmunix: /users: file system full
```

```
Sep 5 13:19:39 tempest vmunix: Ethernet: late collision, check transceiver
Sep 5 14:08:41 tempest vmunix: vnode: table is full
Sep 5 14:08:43 tempest last message repeated 7 times
```

A sample *auth.log* file:

```
Aug 14 10:29:37 tempest Aug 14 10:29:37su: BAD SU math on /dev/ttyp0
Aug 14 10:29:40 tempest Aug 14 10:29:40su: math on /dev/ttyp0
Aug 14 16:37:36 tempest Aug 14 16:37:36su: math on /dev/ttyp1
Aug 21 09:05:25 tempest Aug 21 09:05:25su: angelo on /dev/ttyp1
Aug 21 10:06:02 tempest Aug 21 10:06:02login: 1 LOGIN FAILURE ON ttyp2, angelo
Aug 21 10:15:42 tempest Aug 21 10:15:42login: 1 LOGIN FAILURE ON ttyp2, angelo
```

A sample *lpr.log* file:

```
Aug 09 14:01:36 tempest lpd: lp9: unable to get hostname for remote machine
Aug 09 14:01:40 tempest lpd: gethostbyname 'peking.' Error 0 1 rs (null)
Aug 09 14:18:39 tempest lpd: Printer queue is disabled
Sep 03 14:05:41 tempest lpd[12060]: lpd started
```

Cleaning Up Error Logging Files

If you have **syslogd** configured to create a daily directory as described earlier, **syslogd** automatically creates a new date directory if either of two situations occur: after 24 hours have passed or if the system is rebooted.

The error logging files stored in old directories tend to be short, but the disk space they occupy can become quite large. If all system processes are running without problems, the error logging files are zero-length. Even so, they clutter up the system and should be removed as follows:

```
rm -rf directory
```

Error logging files can also be cleaned up through **crontab**. For example, to delete directories that are more than five days old, enter the following line to the *crontab* file:

```
55 23 * * * find /var/adm/syslog.dated -type d -mtime +5 -exec
rm -rf '{}' ;
```

The preceding **crontab** command line specifies that every day at 23:55, all directories under */var/adm/syslog*.dated that were modified more than 5 days ago should be removed, along with their contents.

Stopping the Error Logging Daemon

After **syslogd** is invoked, it creates the */var/run/syslog.pid* file, in which it stores a single line containing the syslogd process ID. Use this process ID to terminate **syslogd** when you are shutting the system down as follows:

```
kill 'cat /var/run/syslog.pid'
```

To cause a running **syslogd** daemon to take notice of changes in the *syslog.conf* configuration file without shutting down the system, use the following command line:

```
kill -HUP 'cat /var/run/syslog.pid'
```

Printer Services Management 14

The print service software supports a wide variety of printers configured within a remote network. This chapter describes the various directories, files, and programs that configure and manage printer support, which may be extended to large networks. There is no way to directly connect a printer to a Paragon™ system. Printer connections must be made through a network.

This chapter assumes that all of the system printers are remotely connected and that they are fully functional. Refer to printer-specific installation manuals for information on how to remotely connect printers through your remote hosts and to vendor-specific printer hardware manuals for information on how to operate your printers. There is no physical port available to connect a printer directly to the Paragon supercomputer.

Configuration of Printer Services Software

All access to printers is controlled by the **lpd** line printer daemon located in the */usr/sbin* directory, and by the **lpr** line printer program in the */usr/bin* directory.

The **lpr** program puts data into the spooling queue until the printer selected is ready. When the selected printer is ready, the job is passed from the spooling queue to the printer by the **lpd** line printer daemon. To configure printer service software you must first create or edit several directories and files. These files include:

/sbin/init.d/lpd The **lpd** line printer daemon is activated from **lpd** startup file, */sbin/init.d/lpd* file. Printing cannot occur unless the **lpd** line printer daemon is running. The line printer daemon is the spool-area handler. Whenever a user wants to print a file, the **lpr** program sends a copy of the file to the print spooler. The print spooler for a named printer is a FIFO (first-in first-out) queue of all files to be printed.

The file copies remain in the printer spooling queue until the printer is ready to print them. When the printer is ready, **lpd** forks processes that do the actual printing. The **lpd** line printer daemon handles all printing requests, ensures that the requestor has access to the printer, manages the printer spooling queues, and calls the processes that control the printer.

/etc/printcap The master database for the printing system. Every printer in the system must be described in this file. There is a printer configuration group for every printer interfaced with the local host. This printer configuration group specifies all the characteristics about any particular printer that the **lpd** line printer daemon must know.

/dev/printername A directory for printer device reference names, created for each printer whose reference name is specified by the *printername* parameter. The specification is usually **lp***n*, where *n* is an arbitrarily assigned printer number. These files are referenced by the **lpd** line printer daemon.

NOTE

Intel Scalable Systems Division does not maintain nor provide technical support for any listed printers.

/usr/spool/lpd/printername
A subdirectory for the spooled data files queued for printing on each printer specified in the */etc/printcap* file. To be printed, the files are accessed by the **lpr** line printer program. Each spooling directory name is the same as the primary reference name of the printer for which the spooling subdirectory is created.

/etc/hosts.lpd A list of hosts that may access printers interfaced with the local system.

/etc/hosts.equiv A file that contains a list of *trusted* hosts for the print system. A trusted host is a machine having login access to the named host and is also permitted access to the local machine.

The */sbin/rc3* File

The **init** program invokes the */sbin/rc3* file when the system is booted. The */sbin/rc3* file is a shell procedure that performs various functions during start-up, one of which is to start the **lpd** line printer daemon. In most cases, the */sbin/rc3* shell script for your system contains a command line to start **lpd**.

The */etc/printcap* File

The */etc/printcap* file is the descriptive database for printers interfaced remotely with the Paragon supercomputer. You must modify the *printcap* file under *root* authority to explicitly define the characteristics of each printer. Because the Paragon supercomputer has no local printer capability, the */etc/printcap* file must include network configuration information. When the network configuration is to be tightly controlled, security conditions should be met before a user is permitted to spool jobs to a remote printer.

Local Machine Name—The local machine name must be entered into both the */etc/hosts.lpd* and the */etc/hosts.equiv* files on the remote machine. Each user of the printer must have an account on the remote machine (for optional security assurance).

Remote User Access—On the local machine, the only *printcap* file parameter variable you need to set is **rs**, the remote machine access parameter. This a Boolean value that takes on a true (yes) or false (no) state only. When you define the value as true, remote users are required to have a local account before they have access to the local printer. When you define the value as false, any user may access the local printer.

Remote Printer and Machine Names—On the machine that has access to the remote printer, the */etc/printcap* file parameters **rp** and **rm** must both be defined. The **rm** parameter specifies the name of the machine to which the remote printer is physically connected and the **rp** parameter is the name by which the printer is known to the remote machine.

The following example shows typical printer configuration entries for two remotely attached printers in a *printcap* file.

```
lp1|lpb|blake| Blake's postscript laser printer :\
        :mx#0:lp=:rp=lpb:rm=blake:
#
lp3|psf|Frost's NEC laser printer :\
        :mx#0:lp=:rp=lp3:rm=frost:
```

In the foregoing example, each configuration string is also set off with a colon (:). Each entry describes the initial record for the remote machine and printer. The configuration strings for the defined printers are minimal and may not reflect the actual parameters required.

All of the possible printer configuration parameters you may use to specify printer characteristics in each printer configuration group are described in detail in the **printcap(4)** manual page. When you do not specify a parameter, an assigned default value is used. Default values are also described in the **printcap(4)** manual page.

The */etc/printcap* file provides, as a minimum, a printer configuration group for each available printer that is interfaced with the remote hosts, print filters, spooling directories, and the output device associated with each printer. The next several sections describe how to make these printer configuration entries in the */etc/printcap* file. Each configuration string tells the **lpd** line printer daemon what it must know about the printer referenced in the initial entry string. Each configuration string has a leading 2-character mnemonic reference, which tells the **lpd** line printer daemon what parameter is being specified. Up to 42 specifications are possible.

Parameters may be alpha strings, numbers, or Boolean equivalents.

- When a parameter is expressed with alpha characters only, the configuration string must be preceded with an = (equal sign).
- When a parameter is numeric, its configuration string must be preceded by a # (number sign) to indicate a numerical value.
- When a parameter is a Boolean expression, it may only be expressed as true or false.

Each configuration string is also set off with a colon (:), and has the form shown in the previous examples.

In the foregoing example, each configuration string is set off with a colon. Also, each configuration string is ended with a backslash character (\) and carriage return to separate the strings and make reading of the *printcap* file easier. Although this is a helpful practice, it is not necessary. Each of the parameters in the foregoing list defines some specified printer characteristic, names a directory, subdirectory, or file, or specifies some limiting value (an 80-column line length, for example).

Identifying the Name and Number of the Printer

The first two entries required in each */etc/printcap* printer configuration group are the names and device reference designation of each printer. The following example describes a printer named **lp0**:

```
lp0|spiro|local line printer:
```

In the example, `lp0` is the printer reference name for a line printer device that is defined in the */dev* directory and `spiro` is the printer's alternate name. Because **lp0** is normally the primary printer device, this is the first */etc/printcap* file entry for the initial printer configuration group. In an earlier example of the */etc/printcap* file, one of the printers referenced was **lp1** and the alternate name was *blake*.

Any job queued for printing by the **lpr** program that does not define a destination printer is sent to the default printer name. You should assign the reference name **lp** to at least one of the printers.

Defining File and Directory Parameters

Each printer configuration group in the *printcap* file should specify a separate spool directory for each printer. To track potential problems, you should also provide an error log pathname, which is specified by the **lf** configuration string parameter.

Spool Directory—The spool directory pathname for each accessible printer is specified by the **sd** parameter in the *printcap* file. The spool directory pathname for every printer listed in the *printcap* file must have the same parent name, which is usually */var/spool*.

The default spool directory for any printer is */var/spool/lpd*. The following three spool directory specifications for three different printer configuration groups are typical for printers whose names are *purple*, *milhaus*, and *axle*.

```
:sd=/var/spool/lpd/purple:\          spooling directory for purple
:sd=/var/spool/lpd/milhaus:\         spooling directory for milhaus
:sd=/var/spool/lpd/axle:\            spooling directory for axle
```

You must name a spool directory in each printer configuration group for each printer that can be accessed from your system, even when that printer interfaces with another machine or is on another network.

For information about creating spooling directories, see "Spooler Operations Management" on page 14-10.

Error Log Filename—The error log filename for each accessible printer must be specified with the **lf** configuration string parameter. The error log file is normally located in the */var/adm* directory. Although this error log is shared by all printers accessible from the local host, you must specify it for each printer in each *printcap* file configuration group.

The default error log filename for any printer is */dev/console*. The following three error log file specifications for three different printer configuration groups are typical for printers whose names are *purple*, *milhaus*, and *axle*.

```
:lf=/var/adm/lpderrs:\               entry for purple
:lf=/var/adm/lpderrs:\               entry for milhaus
:lf=/var/adm/lpderrs:\               entry for axle
```

As shown in these examples, for every printer accessible through the local host, the error log file you specify should have the same local host pathname in each printer configuration group.

Defining the Accounting File and Line Printer Daemon Filter

To charge users for printing services or to determine how much hard copy that system users are producing, you specify an accounting file. The **pac** program can then write accounting records to that file. You can assign an accounting file name in each printer configuration group with the **af** parameter in */etc/printcap*.

To specify an **lpd** line printer daemon filter directory use the **if** and **of** *printcap* file parameters.

Accounting Filename—Printer accounting is obtained when the **pac** printer accounting command is invoked by the **cron** daemon. You may also invoke printer accounting from the command line, but this is seldom done. Refer to Chapter 15 for a description of printer accounting.

When the **pac** program executes, a record for each instance of printer use is written to an accounting file that is specified in each printer configuration group, together with the amount of paper needed to produce the hard copy. You must specify the accounting file to which these records are written with the **af** parameter of the *printcap* file.

A printer accounting file should be specified only on the machine to which the printer is attached because accounting records are written only to the specified file whenever a job is printed by that printer. The following configuration group entries show typical accounting file specifications for three printers named *purple*, *milhaus*, and *axle*:

```
:af=/var/adm/printer/purple.acct:          accounting file for purple
:af=/var/adm/printer/milhaus.acct:         accounting file for milhaus
:af=/var/adm/printer/axle.acct:            accounting file for axle
```

The following configuration group entries show typical filter program and shell script specifications for three printers named *purple*, *milhaus*, and *axle*:

```
:if=/usr/lib/lpdfilters/ln03rif:           input filter for purple
:of=/usr/lib/lpdfilters/ln03rof:           output filter for purple
:if=/usr/local/lib/iplps/lpif.sh:          input filter for milhaus
:of=/usr/local/lib/iplps/lpof.sh:          output filter for milhaus
:if=/usr/local/lib/iplps/lpif.sh:          input filter for axle
:of=/usr/local/lib/iplps/lpof.sh:          output filter for axle
```

These entries show that filters for the printer named *purple* are located in a different subdirectory from those for the printers named *milhaus* and *axle*. Filter files stored in the */usr/lib* directory are standard operating system support files. Filter files stored in the */usr/local/lib* directory are user support files.

Communications Parameters

You control three communications parameters by establishing their values in the */etc/printcap* file. These parameters include the baud rate, flag bits, and the local-mode bits.

Baud Rate—The baud rate (bits/second), controlled by the **br** configuration string parameter, is the maximum rate at which data can travel between the data source and the printer. Because the baud rate is a numeric value, you must use the number sign (#) to set it. You must also be sure that the printer for which you are establishing a baud rate is capable of receiving data at the specified rate. For information about the baud rate of a specific printer, refer to the appropriate manual for that printer.

Flag Bits—Flag bits are specified with the **fc** and **fs** parameters. A flag bit is a 16-bit octal value that tells the system details about the capability of a particular printer. Not all printers use all the flag bits, but you must either set them or clear them. The states of the flag bits are expressed as octal numbers in a 16-bit word. The flag-bits **Symbolic Name**, **Octal Value**, and **Purpose** are listed in Table 14-1.

The flag bits set particular characteristics of operation or data transmission from the host to the printer and when the printer is capable, from the printer to the host. Data that is passed from the printer to the host could possibly include stop and start status information to tell the host when the printer input buffer is ready to accept input and when its buffer is about to overflow.

The values you use for the **fc** and **fs** string configuration parameters are expressed as octal numbers in Table 14-1 on page 14-8. Octal values are preceded with the numeric **0** (zero) to indicate an octal value. To clear all the bits, for example, you assign the value **0177777** to parameter **fc** and to set all the bits, you assign the value **0177777** to parameter **fs**. Because you may only use a single value for either **fc** or **fs**, you must use the octal sum of the combined bits to set or clear any groups of bits.

The delays are specific times used to delay the transmission of the next group of characters to the input buffer to give the printer mechanism time to perform an operation such as a carriage return, newline, tab, and formfeed. Not all flag bits are used by every printer. You must consult the printer manual for its printer-flag particulars.

Pagination and Imaging Parameters

Printer filters need to know the size of an output page to frame pages and insert linefeeds and carriage returns.

Line Length and Column Width—For line printers, the **pl** and **pw** parameters specify page length in numbers of lines and page or column width in number of constant-width characters.

Image Area Parameters—For high-resolution laser-type printers, the analogous line length and page width parameters are **py** and **px**, which specify the number of pixels along the x- and y-coordinate planes of the printer output image area. Some printers can operate in either constant-width or imaging modes, so you must specify for both modes. For information on output characteristics, consult the printer manual.

Table 14-1. The printcap File Parameters fc and fs, Flag Bits

Symbolic Name	Octal Value	Purpose
TANDEM	0000001	Automatic flow control. In this mode, the printer is capable of producing a stop character when its input buffer is full or a start character when its input buffer can accept data.
CBREAK	0000002	Control break. Passes characters from the input filter to the printer immediately.
LCASE	0000004	Maps uppercase input characters to lowercase. Not normally used with printers.
ECHO	0000010	Echo. Full-duplex operation.
CRMOD	0000020	Outputs a single linefeed as carriage return and linefeed.
RAW	0000040	Raw operating mode. Passes characters from the input filter to the printer immediately.
ODDP	0000100	Odd parity on input.
EVENP	0000200	Even parity on input.
NL1	0000400	Newline delay No. 1.
NL2	0001000	Newline delay No. 2.
TAB1	0002000	Tab delay No 1.
TAB2	0004000	Tab delay No. 2.
CR1	0010000	Carriage return delay No 1.
CR2	0020000	Carriage return delay No. 2.
FF1	0040000	Formfeed delay No. 1
BS1	0100000	Backspace delay No. 1.

Creating Spooling Subdirectories

Each printer must have its own spooling subdirectory located within the */usr/spool/lpd* directory. The spooling subdirectory requires the same name as the full name of the printer it serves. The spooling subdirectory access permissions should be set to **775** with both the owner and group set to the pseudo-user name **daemon**, as shown in the following example:

```
# cd /var/spool/lpd
# mkdir printername
# chgrp daemon printername
# chown daemon printername
# chmod 775 printername
```

In command sequence, *printername* is the reference name of the printer from the first field entry on the first line of the configuration group for the printer in the */etc/printcap* file (see "The /etc/printcap File" on page 14-3).

The following example is a typical file listing entry from the */var/spool/lpd* directory after you create a file named *lp1* and change the owner and group names to **daemon**:

```
drwxrwxr-x 2 daemon daemon 24 Oct 12 1990 lp1/
```

A spool subdirectory is required even when the printer being configured is located on a different machine because all spooled files are locally stored until they can be transferred for printing.

Spool Directory Files

In addition to any files queued for printing, each spool subdirectory contains the files *status* and *lock*. These files are created by the **lpd** printer daemon whenever a file is queued for printing.

/var/spool/lpd/printername/status
: The status file contains a one-line description of the current printer state. This file is maintained by **lpd** and is referenced by the **lpq** command.

/var/spool/lpd/printername/lock
: The *lock* file prevents multiple invocations of the **lpd** line printer daemon and holds information about any active job. The **lpd** program operates on permission bits for the *lock* file to enable spooling and printing.

Controlling Printer Access

The **lpd** printer daemon acknowledges print requests from only those hosts listed in either the */etc/hosts.lpd* file or the */etc/hosts.equiv* file.

NOTE

Intel Scalable Systems Division does not maintain nor provide technical support for any listed printers.

The */etc/hosts.lpd* File

The */etc/hosts.lpd* file is a list that gives one host machine name per line. An asterisk (*) on any line in the */etc/hosts.lpd* file allows print requests from all hosts.

The */etc/hosts.equiv* File

The *hosts.equiv* file is in the */etc* directory and contains a list of trusted hosts. There is no limit to the number of trusted hosts you may name in this file. When an **rlogin** or an **rsh** request is made from any host listed in the *hosts.equiv* file, and the initiator of the request has a permission entry in his local */etc/passwd* file, further validity checking is not required.

Thus, the **rlogin** command process does not prompt for a password, and the **rsh** command process completes successfully. When a remote system is listed in the local *hosts.equiv* file, users of that system are defined as users of the local system with the same user ID. The *hosts.equiv* file provides a list of trusted hostnames, or host and user names. Programs scan the *hosts.equiv* file sequentially.

Spooler Operations Management

This section provides a description of the various printer control programs and explains how to use them.

Once the printer service software is configured, the print spooler requires little maintenance. There will, however, be times when you need direct control of a print job. To have direct control you must understand the print spooling and printer control programs that the system provides. The print service control programs enable you to request printing services from a printer, examine the status of printers and spoolers, control printer activity, and generate printer accounting reports.

The lpr Print Request Command

To print a file, a user executes the **lpr** line printer request command. The syntax of **lpr** is as follows:

lpr [**-hjmrs**] [*filetype_flag*] [**-C** *class*] [**-i**] [*number*] [**-J** *name*] [**-P***printer*] [**-T** *title*] [**-#***number*] [**-1** *font*, **-2** *font*, **-3** *font*, **-4** *font*] [**-w***number*] *file* ... | -

When this print command is processed, the requested printing job is stored in the spooling queue of the local printer and the **lpd** line printer daemon is notified that a file has been queued for printing. When the requested printer is on a remote machine, **lpr** passes a copy of the file to be printed to the spooling queue of the printer that is interfaced with the remote machine. When this cannot be done immediately, **lpr** continues to try to write a copy of the requested file to the printer spooling queue. Whenever a user wishes to print a file, the **lpr** command specifies the name of the destination printer and the name of the file to be printed. When a printer is not named, the file is sent to the default printer spooler.

In addition to the general flags, **-hjmrs**, there are special flags that specify a particular file type. Refer to the **lpr** command manpages.

The following is a typical print command using **lpr**:

```
lpr -Pmilhaus sample.fle
```

The **lpr** process assigns a printer request ID, which is a number that tracks the printing process; it produces control and data files, both of which are identified with an assigned request ID number; and it activates the **lpd** printer daemon. When **lpd** becomes active, **lpd** forks a child process for each call to handle the printing job and continues to listen for other printing requests.

The lpd Printer Daemon

The **lpd** printer daemon uses the system calls **listen**() and **accept**() to transfer file copies to a printer spooling queue, display the printer spooling queue, and remove printed files from the printer spooling queue.

The **lpd** printer daemon scans the */etc/printcap* file to determine the characteristics of the printer whose spooling queue is to be accessed for the next file to be printed. When a system is rebooted, **lpd** sends any files that were not printed when the system last stopped operating to the named printer.

The *lock* file in the spooling subdirectory for each printer prevents the **lpd** daemon from becoming active a second time, and it holds control information about the currently active printing process. When the **lpd** printer daemon is activated, it looks in the printer spooling subdirectory for the presence of a *lock* file. If no *lock* file is present, **lpd** creates one and writes the ID number and the control filename on two successive lines. This file tells the **lpd** printer daemon that the printer is busy, which prevents another printing process from being invoked.

After the **lpd** printer daemon has successfully created a *lock* file, it scans the printer spooling subdirectory for filenames that begin with the characters `cf`. These command files specify the names of other user files that are to be printed and provide parameters that tell **lpd** how queued files should be printed. Each line in a command file begins with a *key* character to indicate what to do with the remainder of the line. The *key* characters and their meanings are described in detail in the **lpd(8)** manual page.

Activating Printer Daemons

The **lpd** printer daemon is the spool directory handler for printers that are interfaced with the local host. Usually, **lpd** is started from the */sbin/rc* file during boot time as a background task and remains running. When not active, **lpd** waits for input from the **lpr** command. For more information, refer to "The /sbin/rc3 File" on page 14-2.

Examining the Printer Spooling Queue

When first invoked, the **lpr** command spawns a child process that looks for the control file, which has `cf` as its first two characters. Printing data files have `df` as the first two characters. These files are identified by their print request ID numbers only. The **cf** files are control files that tell **lpd** how to handle a printing job.

For example, suppose the printer named *milhaus* has printing jobs currently queued in its spooler. A typical method for examining the *milhaus* print spooler is to get into the appropriate printer directory and look at a file listing, as follows:

```
ls -l /usr/var/spool/lpd/milhaus

-rw-rw---- 1   root       75   Oct 17 09:57   cfA0220mothra
-rw-rw---- 1   root       96   Oct 17 10:03   cfA143harald
-rw-rw---- 1   root       96   Oct 17 10:03   cfA144harald
-rw-rw---- 1   root   199719   Oct 17 09:57   dfA0220mothra
-rw-rw---- 1   root     9489   Oct 17 10:03   dfA143harald
-rw-rw---- 1   root     3474   Oct 17 10:03   dfA144harald
-rw-r--r-- 1   root       20   Oct 17 10:06   lock
-rw-rw-rw- 1   daemon    113   Oct 17 10:00   status
```

The control files beginning with the characters `cf` contain printing instructions for each print job. The data files beginning with the characters `df` contain formatted text for printing. The *lock* file contains the process ID of the daemon that is currently running and the status file contains a line describing the current printer status. The content of this line is printed whenever a user makes a status inquiry about *milhaus*. When any printer whose status is queried is not active, the status message written to standard output is `no entries`.

Controlling Printer Spooling Queues

Two commands control printer spooling queues: **lprm** and **lpc**. The **lprm** command may be called by any user. The **lpc** command, however, requires that the caller have superuser authority.

The **lprm** command permits users to remove printing jobs queue in a particular spooling directory. The **lpc** command is used by a system administrator to control the operation of the printing system.

Removing Printing Jobs from a Spooling Queue

The **lprm** command deletes queued printing jobs from a printer spooling subdirectory. When necessary, **lprm** stops a running line printer daemon that is currently processing the contents of the referenced spooling queue. After the specified files have been removed from the spooling queue, this command restarts the **lpd** line printer daemon.

When you call **lprm** without specifying parameters, the currently active printing job is deleted from the default printer spooling queue if this command is called by the user who called the **lpr** line printer request command that put the data file into the spooling queue. Generating Printer Accounting Reports

The printer daemon **lpd** records printer usage when the */etc/printcap* file has the **af** parameter specified and the accounting file specified by the **af** parameter has been created in the */var/adm* directory. Each printer should have its own accounting file. Accounting is discussed in detail in Chapter 15.

Accounting 15

In multi-user environments that are under the control of multitasking operating systems, system resources are time-shared by all users of the system. The accounting subsystem allows you to track system resources. With the accounting subsystem, you obtain a diagnostic history of system resource use as well as a history of user activity. User activity-logging permits direct charges to be made against individual uses of machine time and system resources.

Although you can structure your own complete accounting system with the provided accounting commands, the various accounting shell scripts provide a means to invoke automatic system accounting on a daily basis. To enable automatic accounting, you must first set up your **cron** daemon to call the various accounting shell scripts on a periodic basis that meets your needs.

Accounting Overview

When accounting is enabled, various database files are automatically maintained by the kernel and accounting system processes. The kernel and accounting system processes automatically write records to a number of database files, which are the source of all accounting information. When a process exits, the kernel writes process accounting records to the */var/adm/pacct* database file. These records contain a UID, a command name, and information about resource usage, such as the amount of CPU time used by the process and the number of characters and blocks read and written. The **init** and **login** programs write connect-time records to the */var/adm/wtmp* database file when users log in and log out. From these two sets of records, commands in the accounting subsystem can determine when users log in and log out and what system resources they used (such as disk space, printing services, and CPU time).

The commands in the accounting subsystem are designed to further automate the accounting process. For example, the **runacct** script is usually run once a day through **cron**. This script calls a number of accounting commands that read the *pacct* and *wtmp* files, interpret the records, and produce files that summarize and sort the information in various ways. Still other commands produce various kinds of resource-use and user-history reports from these summary files. These commands can also be run through **cron** at appropriate intervals to provide information for archives, for diagnostics, for resource billing, or for reporting to management.

Resource Accounting

The use of system resources can be tracked on an almost continual basis. The accounting database files are organized by connect-sessions, which begin when a user logs in and end when that user logs out. During a connect-session, process-accounting processes monitor CPU time usage (consisting of the amount of direct CPU time used out of the total user connect time), memory usage, I/O transactions, and the number of commands and scripts called by the user. Also monitored are disk and printer usage.

When system resources use is tracked by accounting processes, distinctions are made between resource access during peak and nonpeak hours. Peak hours are called *prime-time*; other hours, including weekends and holidays, are called *nonprime-time*. Both the prime-time hours and holidays are defined in the */etc/acct/holidays* file and a sample file is included with the system. This file generally requires editing to reflect the holidays your organization observes. It may require editing once a year if the dates of these holidays change.

Connect-Session Processes

The **login** and **init** programs record connect-session times by writing records to the */var/adm/wtmp* file. This file is written every time a user logs in or out. The *wtmp* file has a fixed format; each record provides the following information:

- User login name from the */etc/passwd* file
- User ID from the */etc/inittab* ID file
- The device name (*console* or *tty23*, for example)
- Type of entry
- Process ID number
- Process termination status
- Process exit status
- Time entry was made
- Host machine name

Binary records are continuously written to *wtmp*, so it can become excessively large. Consequently, *wtmp* should be periodically truncated (monthly, for example), depending on how fast it grows. How often you do this should be established as a periodic accounting task that you can perform manually or through a script that you write to control *wtmp* file length. Using one of the accounting commands, it is also possible to edit the *wtmp* file.

Process-Accounting Processes

After automatic system accounting is installed and enabled, process accounting takes place whenever a command, shell script, or program is executed anywhere in the system. Whenever a process terminates, the kernel writes a process accounting record to the daily */var/adm/pacct* file. The *pacct* file has a fixed format and each record provides the following information:

- An accounting flag that qualifies the process (a child process, for example)
- Exit status that indicates how the process terminated
- User ID of this process
- Group ID of this process
- Terminal from which process originated
- Process start time
- User time used by process
- System time used by process
- The amount of time (CPU) used by the process
- The amount of memory used by the process
- The number of (I/O) characters transferred by the process
- The number of (1024-byte) blocks read or written by the process
- The name of the command used to start the process

The process accounting records provide the means to monitor program execution statistics on a periodic basis. Records are written to the default process accounting database file */var/adm/pacct*, but you can name some other file as the process accounting database. When you name some other file as the process accounting database, that file must exist.

As with the *wtmp* file, the *pacct* file can grow to an excessively large size. A command in the accounting subsystem, which is usually run through **cron**, ensures that when the *pacct* file becomes too large, another *pacct* file is created.

Disk-Usage Accounting Processes

Disk usage accounting is performed by the **/usr/sbin/acct/dodisk** script whenever it is invoked (normally from the **cron** daemon at some periodic interval). When the **dodisk** command is executed, binary disk usage information is written to the */var/adm/acct/nite/daytacct* file whenever disk usage has been detected. By default, disk accounting is done on the file systems specified in the */etc/fstab* file, which contains a list of mountable file systems.

The content of the */var/adm/acct/nite/daytacct* file can be combined with other system resource accounting information, or it can be directed to a file for reporting using various accounting commands or scripts.

Printer-Usage Accounting Processes

Printer usage accounting is obtained when the **/usr/sbin/pac** command is called, either from the standard input or from a script. If a printer's name is specified in the */etc/printcap* file and its accounting feature is enabled, a record for every print job is written to a printer accounting file in the */var/adm/printer* subdirectory. These printer accounting records have a fixed format that provides the following information:

- Host name and user name
- Number of pages or feet of medium printed
- The number of times the printer was used
- The price per unit of printed output

The printer accounting records provide the means by which users can be charged appropriately for system printing resources.

Accounting Commands and Shell Scripts

There are 20 commands and 14 shell scripts in the accounting command group. The difference between a command and a shell script is normally transparent to the user. A command is an individual entity that is usually designed to accomplish a specific task, while a shell script may contain a sequence of several different commands.

Accounting shell scripts write records to various accounting database files. Sometimes a shell script creates an accounting database and sometimes the system administrator does. It depends on the application. The simplest way to customize the accounting subsystem is to copy an existing script and modify it so that it calls commands with different flags or writes to different files. The accounting commands and scripts are listed in Table 15-1.

Table 15-1. Accounting Commands and Scripts (1 of 2)

Name	Type	Purpose
/usr/sbin/ac	Command	Prints connect-time records to the standard output.
/usr/sbin/pac	Command	Prints printer accounting records to the standard output for each user who outputs hard copy on a printer.
/usr/sbin/sa	Command	Summarizes accounting records and prints them to the standard output.
/usr/bin/acctcom	Command	Prints process accounting records in the */var/adm/pacct* (default) or from some other specified binary accounting database file.
/usr/bin/last	Command	Prints previous login information to the standard output.
/usr/bin/lastcomm	Command	Prints last commands information to the standard output.
/usr/sbin/acct	Directory	Accounting command directory. This directory holds accounting scripts and programs.
chargefee	Script	Writes your charge-fee record to */var/adm/fee*, the charge-fee database for any time you use to perform some service to system users.
ckpacct	Script	Checks the size of binary process accounting file */var/adm/pacct* for excessive record content.
dodisk	Script	Writes daily disk usage accounting records to disk usage accounting file */var/adm/acct/nite/daytacct*.
lastlogin	Script	Writes the date of the last login for all users to */var/adm/acct/sum/loginlog*.
monacct	Script	Writes the daily binary *tacct.mmdd* summary files (for the month and day specified by *mm* and *dd*, respectively) from subdirectory */var/adm/acct/sum* (as monthly *tacct.mmdd* summary files) to subdirectory */var/adm/acct/fiscal*. A new daily *tacct* binary summary file is then started in subdirectory */var/adm/acct/sum* for the current day.
nulladm	Script	Creates accounting files and sets permissions (modes).
prctmp	Script	Prints */var/adm/acct/nite/ctmp*, the connect-session record file, to the standard output.
prdaily	Script	Collects daily accounting records from various files and writes them to the standard output.
prtacct	Script	Formats and prints in ASCII any binary *tacct* daily accounting file in subdirectories */var/adm/acct/nite* or */var/adm/acct/sum* to the standard output.

Table 15-1. Accounting Commands and Scripts (2 of 2)

Name	Type	Purpose
remove	Script	Removes any *wtmp* and *pacct* accounting files in subdirectory */var/adm/acct/sum* and the lock file in subdirectory */var/adm/acct/nite*.
shutacct	Script	Turns off accounting.
startup	Script	Enables accounting processes.
turnacct	Script	Controls creation of process accounting files.
acctcms	Command	Prints formatted command usage summaries from binary command summary file */var/adm/acct/sum/cms* to the standard output.
acctcon1	Command	Summarizes content of */var/adm/wtmp* file records into one line per connect session ASCII records.
acctcon2	Command	Summarizes content of files formatted by the **acctcon1** command.
acctdisk	Command	Provides overall disk usage accounting.
acctdusg	Command	Provides disk block usage accounting.
acctmerg	Command	Merges records and writes them to the standard output or to a specified file.
acctprc1	Command	Prints type `acct` structure records by user ID and login name.
acctprc2	Command	Prints type `acct` structure records by user ID and real name.
accton	Command	Permits superuser to control when records are to be written to an accounting file.
diskusg	Command	Provides disk accounting data by user ID.
fwtmp	Command	Prints the */var/adm/wtmp* binary file records.
acctwtmp	Command	Writes records to the */var/adm/wtmp* file.
wtmpfix	Command	Corrects date and timestamp inconsistencies in */var/adm/wtmp* formatted files.
printpw	Command	Prints the content of password database file */etc/passwd* to the standard output.
runacct	Script	Runs daily accounting processes. This file calls various accounting programs and scripts to write information to various accounting files on a daily basis.

As noted in the descriptions in the **Purpose** column of Table 15-1, many files are associated with each of the commands and shell scripts. For example, the principle accounting database files are the *wtmp* and *pacct* files located in the */var/adm* subdirectory. With the exception of the *wtmp* file, which you must initially create, most of the other accounting files are produced and written by the kernel or the **runacct** script, which calls accounting commands or other scripts. The **runacct** process is called by the **cron** daemon and is normally run on a daily basis during nonprime-time hours.

Some of these files often become quite large during day-to-day accounting operations. The files that tend to become large are usually written in binary format to optimize performance when they are read. Others are written and maintained as ASCII files.

Accounting Subdirectories and Files

The subdirectories and files associated with accounting commands and scripts are listed in Table 15-2. The **Name** column lists the subdirectory name or the filename. The pathname of any file listed under a subdirectory heading must have both the subdirectory name and filename. The **Type** column tells you whether the entry is a directory, an ASCII file, or a binary file. The **Purpose** column tells you what the entry is used for. Table 15-2 is also partitioned according to the directory holding a particular group of files.

Extraneous Files

Files other than those listed in the preceding tables are produced by the accounting commands and scripts, but for the most part these files are only temporary and exist only while a process is running. Under some circumstances (when a process terminates prematurely, for example), one or more of these temporary files can appear in one of the subdirectories listed in the table. Consequently, you should check these subdirectories at some periodic interval, and remove unwanted files.

Corrupted or Lost Files

Accounting files can become corrupted or lost. Many lost or corrupted files can be disregarded. Lost files needed to produce daily or monthly reports can be recovered from backup archives. Other files, such as */var/adm/wtmp* and */var/adm/acct/sum/tacct*, must have complete integrity because they provide the accounting database. Accounting commands permit you to repair such files.

Setting Up an Accounting Schedule

The accounting commands and scripts permit you to monitor the use of many system resources that are controlled by the operating system. Any schedule for an accounting system depends on what you want to do with the available information. As system administrator, you have the option of obtaining a comprehensive quantity of information. You can also turn off accounting completely and truncate any daily files that were previously produced.

Table 15-2. Accounting Subdirectories and Files (1 of 4)

Name	Type	Purpose
/var/adm	Directory	Stores accounting database files and miscellaneous administration files.
diskdiag	ASCII	Diagnostic output produced by the disk accounting command **dodisk**.
dtmp	ASCII	Temporary output produced by the **dodisk** command.
fee	ASCII	Stores output from **/usr/sbin/acct/chargefee** script when you enter the **chargefee** command.
pacct	Binary	The cumulative process accounting database file. Whenever a process terminates, process information is written to this file.
pacctn	Binary	Alternate *pacct* file. A single *pacct* file is limited to no more than 500 1024-byte disk blocks. The size of these files is monitored by the **runacct** script. Each time a new *pacct* file is created, the value *n* is incremented.
Spacctn.mmdd	Binary	Special pacct files produced by the **runacct** script for the month and day specified by *mm* and *dd*, respectively.
lineuse	ASCII	Terminal (tty) line connect time. This file provides line use statistics for each terminal line used during the accounting period.
reboots	ASCII	List of system reboots during accounting period.
wtmp	Binary	The cumulative login/logout accounting database file. Whenever a user logs in or out, connect-time and user information is written to this file.
usracct	Binary	Used by the **sa** command to store per user process accounting summary records.
qacct	ASCII	Contains printer accounting records and is used by the **runacct** script.
savacct	Binary	The **sa** command uses this file to store system process accounting summary records.
/var/adm/printer	Subdirectory	Stores printer user and summary accounting files.
printer.acct	ASCII	Printer user file. This file lists the amount and cost of print media (pages or feet of output media) used for all printers specified by printer according to machine and user login name for all printers for which accounting is enabled.

Table 15-2. Accounting Subdirectories and Files (2 of 4)

Name	Type	Purpose
printer.acct_sum	ASCII	Printer summary files. Each file lists a total summary of media produced by each printer specified by printer according to machine and login name for which summary accounting is enabled.
/var/adm/acct/fiscal	Subdirectory	Stores monthly files and reports.
cms.mmdd	Binary	Active monthly command summary file for the month and day specified by *mm* and *dd*.
fiscrpt.mmdd	ASCII	Monthly accounting report for the month and day specified by *mm* and *dd*.
tacct.mmdd	Binary	Cumulative total monthly accounting file; updated on a monthly basis by the **runacct** script.
/var/adm/acct/sum	Subdirectory	Stores daily summary files and reports whenever the **monacct** command is executed.
cms	Binary	Active total monthly command summary file. Whenever the **runacct** script is executed, records are written to this file to obtain the total command summary file in the internal command summary format.
daycms	Binary	Monthly previous day command summary file. Whenever the **runacct** script is executed, monthly command summary records for the previous day are written to this file.
loginlog	ASCII	Last login date. This file lists the last login date for each user on the system.
tacct	Binary	Cumulative daily accounting file. This file is the total daily accounting file for system use. It is updated on a daily basis by the **runacct** script.
tacct.mmdd	Binary	Daily total accounting file. This file is the daily total accounting file for the month and day specified by *mm* and *dd*.
tacctprev	Binary	Previous *tacct* file. This is the same as the *tacct* binary file, but the one for the previous 24-hour accounting period.
rprt.mmdd	ASCII	Daily accounting report for the month and day specified by *mm* and *dd*.
/var/adm/acct/nite	Subdirectory	Stores daily accounting diagnostic, process-accounting, and connect-session record files.
accterr	ASCII	Diagnostic output from **runacct** script.

Table 15-2. Accounting Subdirectories and Files (3 of 4)

Name	Type	Purpose
active	ASCII	Daily **runacct** process progress file. As the **runacct** script executes, it writes information about its progress to this file. This file also contains error and warning messages.
activemmdd	ASCII	Daily **runacct** error file for the month and day specified by *mm* and *dd*. Same as the active file after an error has been detected by the **runacct** script.
cms	ASCII	Active total daily command summary file. This is the ASCII version of the */var/adm/acct/sum/cms* file. This file is an output of the **prdaily** accounting script, which is called by the **runacct** script to rewrite the */var/adm/acct/sum/cms* file records.
ctacct.mmdd	Binary	Total accounting records derived from connect-session accounting records for the month and day specified by *mm* and *dd*, respectively. This is a temporary file that is deleted each accounting period after the */var/adm/acct/nite/daytacct* file records are written.
ctmp	ASCII	Temporary login/logout record file. This file is the output of the **acctcon1** accounting command, which is called by the **runacct** script to rewrite the */var/adm/wtmp* file records.
daycms	ASCII	Monthly previous day command summary file. This is the ASCII version of the */var/adm/acct/sum/daycms* file. This file is an output of the **prdaily** script, which is called by the **runacct** script to rewrite the */var/adm/acct/sum/daycms* file records.
daytacct	Binary	Total accounting records in the *tacct* format for the previous day. These records are produced when the **prtacct** command is called by the **prdaily** script. The **prdaily** script is called by the **runacct** script when the **cron** daemon runs the daily accounting processes.
dacct	Binary	Weekly total disk usage accounting records in the *tacct* format when the **acctdisk** command is called by the **dodisk** script.
lastdate	ASCII	This file specifies the last day that **runacct** was executed.
log	ASCII	Diagnostic output from the **acctcon1** command.
owtmp	Binary	Daily *wtmp* file after correction produced by **wtmpfix** command.

Table 15-2. Accounting Subdirectories and Files (4 of 4)

Name	Type	Purpose
ptacctn.mmdd	Binary	Additional daily *pacct* files for the month and day specified by *mm* and *dd*. These files are created when the daily *pacct* file requires more than 500 blocks of disk space.
statefile	Binary	Current state of **runacct** script during execution.
wtmp.mmdd	Binary	Daily login/logout accounting database file for the month and day specified by *mm* and *dd*, respectively. Connect-session records about users who logged into the system during the previous day are written to this file.
wtmperror	ASCII	Error message produced when a *wtmp* file is fixed during execution of the **wtmpfix** command.
wtmperrormmdd	ASCII	Same as *wtmp*error after the runacct script detects an error during execution of the **wtmpfix** command for the month and day specified by *mm* and *dd*.
lock	ASCII	Used to assure that only one copy of **runacct** is running at the same time. This file is removed when the **runacct** script completes.
cklock	ASCII	Used to assure that only one copy of **ckpacct** is running at the same time. This file is removed when the **ckpacct** script completes.

A Complete Accounting Schedule

A complete accounting schedule provides information about user login and logout according to hostname, username, or user ID; total connect time; CPU time; memory usage; the number of I/O operations and the number of characters transferred; disk space usage by blocks; printer usage according to printer name, number of printing operations, the amount of printed matter together with a printing charge rate; and modem usage.

All system resource usage takes place during prime time or nonprime time. You can specify any prime-time period you wish in the */etc/holidays* file. A typical complete accounting schedule is suggested by the following list, which can be keyed according to hostname, username, user or group ID, or terminal type.

- Prime connect time
- Nonprime connect time
- Prime-time CPU usage

- Non prime-time CPU usage
- Number of processes spawned
- Number of connect sessions
- Memory usage on a daily, weekly, monthly, or other periodic interval
- Disk usage on a daily, weekly, monthly, or other periodic interval
- Printer usage by username or device type (such as a line printer, laser printer, or phototypesetter)
- Amount of printed media produced
- Modem use and phone connect time
- Service charges

To provide separate accounting procedures for each machine or workstation in a networked environment and then to collect the information can become a huge task. As system administrator, you should design the accounting system that tracks resource usage for all terminals or workstations from a single host in a system. To accomplish this you must design scripts whose command and shell calls provide you with the accounting information you need to gather at regular intervals. Commands and calls to scripts can be written into the */usr/spool/cron/crontabs/adm* file for automated accounting. This file tells the **cron** daemon when to run the processes you want executed.

Rate Schedules

Use of all of the resources listed in "A Complete Accounting Schedule" on page 15-11 can be charged at any rate you determine. The granularity of your rate schedule can be refined to almost any degree. A simple rate schedule, for example, might be a fixed charge per user per minutes of total connect time. This rate could be further refined to include another rate for minutes of CPU usage. Variations are virtually endless.

Although the accounting subsystem does not determine rates for services, it does provide the **/usr/sbin/acct/chargefee** script. For example, if a user inadvertently erases the wrong file, its backup must be rewritten to the subdirectory from which it was deleted. When you must charge for such services, you can use the **chargefee** script to charge any user a given number of charge units. The value of these units is not fixed so that they can have any value you assign to them. These charge units are then collected in the *fee* file in the */var/adm* subdirectory.

File and Report Backups

Many of the accounting files and reports listed in Table 15-2 on page 15-8, which you or the accounting system can produce as part of your system accounting activity, must be periodically truncated or they will eventually fill all of the available disk space. However, before you truncate a file or report you should determine whether or not it needs to be archived.

Just which files and reports you want archived is a matter of assessing the importance of the information contained in them. You could possibly back up every accounting file every day. On the other hand, you may not want to back up any of them. Refer to Chapter 7 for information about a backup schedule for accounting files and reports.

Periodic Accounting Operations

Periodic accounting operations are usually set to take place during nonprime-time hours. Whenever a system is booted, the system initialization shell procedure **rc** must execute the **/usr/sbin/startup** accounting shell procedure. The **startup** shell script initializes the accounting process with the following tasks ("Setting Up Accounting Files" on page 15-14 provides instructions for setting up automatic accounting):

1. The boot process activates accounting, which writes a boot record to the */var/adm/wtmp* file.
2. Process accounting is turned on, which creates the daily */var/adm/pacct* file.
3. Working files previously written to the */usr/adm/acct/sum* subdirectory are deleted and replaced with new ones.
4. If summary files exist, a report for previous day accounting information is created as */var/adm/acct/sum/rprt.mmdd*.
5. The **/usr/sbin/acct/ckpacct** command runs at hourly intervals to ensure that the */var/adm/pacct* file does not grow excessively large. When this file is too large, a *pacctn* file is created. The new file is then written with *pacct* records and monitored for similar growth.
6. The **/usr/sbin/acct/runacct** script runs each night to process the active daily */var/adm/fee* file, the */var/adm/wtmp* connect-session file, the */var/adm/acct/nite/daytacct* disk-usage file, and the */var/adm/pacct* process-accounting file. The **runacct** script also produces command and resource-usage summaries in the */var/adm/acct/sum* subdirectory.
7. The **/usr/sbin/acct/dodisk** command runs once a week to monitor disk usage.
8. The **usr/sbin/acct/monacct** script runs once a month and summarizes the daily or weekly summary files.

The **ckpacct**, **runacct**, **dodisk**, and **monacct** commands are run by **cron**. Depending on the requirements for your system, you can also run other accounting commands, send reports to a printer, send mail to users, and so on, through **cron**.

Setting Up Accounting Files

To set up automatic accounting, perform the following steps:

1. Create the required accounting subdirectories if they do not exist.
2. Create an accounting administration login in the */etc/passwd* file if it does not exist.
3. Set the *PATH* environment variable for the accounting login so that the necessary accounting commands can be accessed.
4. To start accounting, edit the system startup and shutdown files so that accounting begins when the system boots and stops when the system is shut down.
5. Set up the */usr/spool/cron/crontabs/adm* file to call the periodic automatic accounting procedures you want executed.
6. Edit the */etc/holidays* file to provide a prime-time reference.
7. Create the */var/adm/wtmp* file.
8. To enable printing accounting, set up the */etc/printcap* file and the **cron** daemon to automatically produce printer accounting files.

Resource Accounting Directories and Files

The next several sections introduce the directories and files that provide the accounting services that enable you to track system resource usage.

The */etc/passwd* File

For most systems, accounting procedures are not set up to run by default. Consequently, you must be sure that an **adm** login exists by checking the */etc/passwd* file for an entry similar to the following:

```
adm:*:4:4:Administrative Login:/usr/adm:/bin/sh
```

If this entry in the */etc/passwd* file does not exist, add it. Traditionally, the **adm** login has the UID value 4, but this remains a matter of choice. Be sure that your use of the UID value is consistent.

Accounting Subdirectories

Automatic accounting requires three subdirectories. Make sure that the following directories exist, and create them if they do not:

/var/adm/acct/nite
/var/adm/acct/sum
/var/adm/acct/fiscal

These subdirectories must be owned by the user **adm**, who must have the necessary permissions to create and delete files in them.

The */usr/adm/.profile* File

To ensure that the system knows where to find all the accounting commands, set the *PATH* variable in the */usr/adm/.profile* file as follows:

```
PATH=/usr/sbin/acct:/bin:/usr/bin:/sbin:/usr/adm
```

`/usr/adm` is the home directory of the user **adm** as specified in the */etc/passwd* entry.

The */var/adm/wtmp* and */etc/utmp* Files

The **init** and **login** commands maintain the two system login files named *utmp* in the */etc* subdirectory and *wtmp* in the */var/adm* subdirectory. The **login** command records all active logins and logouts in the *utmp* file and accumulates the user login/logout history in the *wtmp* file. The *utmp* file is created by the **init** command. Create the *wtmp* file with the **touch** command as follows:

```
touch /var/adm/wtmp
```

The *wtmp* file should be owned by **root**.

The */var/adm/nite/statefile* File

The **runacct** script uses the *statefile* for temporary storage, but does not create it. Create this file as follows:

```
touch /var/adm/nite/statefile
```

The *statefile* file should be owned by **adm**.

The Startup and Shutdown Files

The */sbin/init.d/adm* script contains commands that enable accounting when the system starts up and end it gracefully when the system shuts down. Make links to this file in the */sbin/rc0.d* directory and the */sbin/rc2.d* directory. See "System Initialization Files" on page 5-1 for more information about these scripts and directories.

The */usr/spool/cron/crontabs/adm* File

To automatically invoke daily and monthly accounting procedures, you must create a */usr/spool/cron/crontabs/adm* file so that the **cron** daemon runs accounting commands and scripts at appropriate times. If you are not familiar with the format of **crontab** files, see the **crontab** manual page in the *OSF/1 Network Application Programmer's Guide*. To set up the */usr/spool/cron/crontabs/adm* file, perform the following steps:

1. Log in as **root**.

2. If it does not already exist, create the */usr/spool/cron/crontabs/adm* file and edit it so that it contains the accounting commands you want executed at periodic intervals.

3. Enter a command to run the **/usr/sbin/acct/runacct** script daily at some nonprime-time hour when most users are not expected to be logged in. For example:

   ```
   0 2 * * 1-6 /usr/sbin/acct/runacct 2>/var/adm/acct/nite/accterr
   ```

   This entry executes the **runacct** script at 2:00 a.m. on Monday through Saturday and redirects any error messages to the */var/adm/acct/nite/accterr* file.

4. Enter a command to run the **/usr/sbin/acct/ckpacct** script hourly to check the length of the */var/adm/pacct* process accounting file. For example:

   ```
   5 * * * * /usr/sbin/acct/ckpacct
   ```

   This entry runs **/usr/sbin/acct/ckpacct** script every hour at 5 minutes past the hour.

5. Enter a command to run the **/usr/sbin/acct/monacct** script monthly at some nonprime-time hour when most users are not expected to be logged in. For example:

   ```
   0 4 1 * * /usr/sbin/acct/monacct
   ```

   This entry runs the **monacct** script at 4:00 A.M. on the first day of the month.

The following example shows how the */usr/spool/cron/crontabs/adm* file might look after you have made the proper entries.

```
#   Entered on 26.05.92 by N. A. Furt
#
#   Adm file for daily, weekly, and monthly accounting
#   procedures
#
#   Daily accounting:
#
#   Check the length of the accounting files
5 * * * * /usr/sbin/acct/ckpacct
#
#   Starting the "ckpacct" command 5 minutes past the hour
#   prevents any of its processes from colliding with other
#   processes previously started on the hour that have not
#   finished.
#
#   Run the daily accounting processes
0 2 * * 1-6 /usr/sbin/acct/runacct 2>/usr/adm/acct/nite/accterr
#
#   To gain access to all files, weekly disk accounting is
#   done from the "root" file in the "/usr/spool/cron/crontabs"
#   subdirectory.
#
#   Do monthly accounting:
0 4 1 * * /usr/sbin/acct/monacct
#
```

The explanation of automatic accounting in this chapter assumes that these three commands are run at these intervals. You will probably want to add more entries to generate various reports and to perform housekeeping functions such as rotating the *wtmp* file.

The */usr/spool/cron/crontabs/root* File

Edit the */usr/spool/cron/crontabs/root* file and add an entry to run the **/usr/sbin/acct/dodisk** script at weekly intervals at some non-prime-time hour when most users are not expected to be logged on. For example:

```
0 3 * * 4 /usr/sbin/acct/dodisk 2>/var/adm/diskdiag
```

This entry runs **dodisk** at 3:00 A.M. on Wednesdays. Because the **dodisk** process must be able to access all files in the system, it should be run under **root** authority.

The */etc/holidays* File

The */etc/holidays* file defines when prime and non-prime time occur, and when holidays (which are always non-prime time) must be counted. The following example is typical:

```
* Prime/Nonprime Table for 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
   43       Feb 19      Washington's Birthday
   76       Mar 17      Saint Patrick Day
  138       May 28      Memorial Day
  185       Jul 4       Independence Day
  249       Sep 3       Labor Day
  319       Nov 22      Thanksgiving Day
  320       Nov 23      Day after Thanksgiving
  358       Dec 24      Day before Christmas
  359       Dec 25      Christmas day
```

Lines beginning with an asterisk (*) are treated as comments. Fields in this file can be separated by any number of spaces or tabs.

The first non-comment line defines the year, the hour at which prime time starts, and the hour at which it ends. Each of the other lines identifies one holiday. The first field of a holiday line is the number of days since the beginning of the current year. The second and third fields express the month as a 3-character expression and the day of the month as an integer. The fourth field is ignored by processes accessing records in this file.

Printer Accounting

Printer accounting is controlled by line printer daemon **lpd**. When the */etc/printcap* file contains a definition for the **af** capability, which is the name of a specific printer accounting file, printer accounting is enabled for that printer.

There are two formats for printer accounting files: the *printer user* format and the *printer summary* format. The printer user format lists printer use according to the user login name and the machine from which the print request originated. The printer summary file lists the amount of media (number of printed pages or feet of roll paper or film) produced by each printer specified with the **pac** command.

You must do the following for each printer listed in the *printcap* file whose use you wish to monitor:

1. Edit the */etc/printcap* file and create or modify the entry for the printer to be monitored.

2. Create a file in the */var/adm/printer* subdirectory whose name is the same as the one listed for the **af** capability.

3. When automatic accounting is desired, enter a printer user accounting command in the */usr/spool/cron/crontabs/adm* file. The printer accounting command **pac** provides a selectable output format for the printer user file.

4. When automatic accounting is not desired, you can enter a **pac** printer user command at the command line anytime you want to want to view the records.

5. When a printer use accounting summary file is also required, create a file in the */var/adm/printer* subdirectory whose name is the same as the one listed for the **af** capability.

The printcap File

The */etc/printcap* file is a database that describes particulars about each printer available to the system. The *printcap* file has one entry for each printer describing the printing features that the printer spooling daemon must know about. Any change made to this file immediately affects printer spooling unless that printer is currently active. See the **printcap** manual page for a definition of the *printcap* file variables. For example, to specify printer accounting for a printer named *liddy*, add the following line to its entry in the *printcap* file:

```
:af=/var/adm/liddy.acct:
```

`liddy.acct` is the name you have given to the printer accounting file for the printer named *liddy*. Printer accounting files must have the extension **.acct**.

The following example is typical:

```
liddy|lps20|LPS20 PrintServer|CF Lounge|os|:\
:lp=@liddy.os.org/print-srv:\
:pl#66:pw#80:mx#0:e :sd=/usr/spool/lpd/liddy:\
:sb:lf=/usr/adm/liddy.log:\
:of=/usr/local/lib/iplps/lpof.sh:\
:if=/usr/local/lib/iplps/lpif.sh:\
:af=/usr/adm/printer/liddy.acct:
```

The *printcap* file and the specific printer parameters that can be specified for a given printer are described in detail in Chapter 14.

Printer Summary File

The printer-summary accounting file is similar to the printer-use accounting file except that it defines the amount of media (pages or feet of paper) output by a named printer. You also use the **pac** printer command to obtain a printer-account summary file, but with different flags. You must create the file to which the printer summary records are written in the */var/adm/printer* subdirectory. This file must have the same name as the printer name defined with the *printcap* file variable **af** and the extension *.sum_acct*.

For example, suppose that your entry in the *printcap* file for the printer named *liddy* is as follows:

```
:af=/usr/adm/printer/liddy.acct:
```

The proper name for the accounting summary file you must create for the printer named *liddy* in the */usr/adm/printer* subdirectory is *liddy.sum_acct*.

This printer accounting summary file should probably be produced no more than once a month.

Automatic Printer Accounting

You can use automatic printer accounting to write the printer-user and printer-summary files, or you can enter the **pac** command at the command line. For automatic printer accounting, enter a line similar to the following one in the */usr/spool/cron/crontabs/adm* file:

```
30 2 1 * * /usr/sbin/pac
```

The preceding entry tells **cron** to run the **pac** command at 30 minutes past 2:00 a.m. each day.

To obtain a monthly printer summary, enter a line similar to the following one for each of the printers in your system for which you want accounting information in the */usr/spool/cron/crontabs/adm* file:

```
30 2 1 * /usr/sbin/pac -Pliddy -s
```

The **-P** flag specifies a printer name, `liddy` in this example, and the **-s** flag specifies a printer accounting summary file. The summary file is created at 2:30 a.m. on the first day of the month.

For example, after adding lines similar to the preceding ones, the typical */usr/spool/cron/crontabs/adm* file in "The /usr/spool/cron/crontabs/adm File" on page 15-16, looks like this when you want printer user and summary accounting files to be produced automatically:

```
# Entered on 26.05.92 by T. A. Trehane
#
# Adm file for daily, weekly, and monthly accounting
# procedures.
#
# Do daily accounting:
```

```
#
# Check the length of the accounting files.
5 * * * * /usr/sbin/acct/ckpacct
#
# Starting the "ckpacct" command 5 minutes past the hour prevents
# any of its processes from colliding with other processes
# previously started on the hour that have not finished.
#
# Run the daily accounting processes.
0 2 * * 1-6 /usr/sbin/acct/runacct 2>/usr/adm/acct/nite/accterr
30 2 * * * /usr/sbin/pac
#
# To gain access to all files, weekly disk accounting is done
# from the "root" file in the "/usr/spool/cron/crontabs"
# subdirectory.
#
# Do monthly system accounting:
0 4 1 * * /usr/sbin/acct/monacct
#
# Do printer summanry accounting for specified system printers.
30 2 1 * * /usr/sbin/pac -Pliddy -s
#
```

In the preceding example, the printer user accounting facet of the **pac** command has been added to the `Do daily accounting` entry and the printer summary facet of the **pac** command has been added to the `Do monthly accounting` entry.

Monitoring System Activity

When you have set up automatic accounting as described in "Setting Up Accounting Files" on page 15-14, the **runacct** script, the **dodisk** script, and the **monacct** scripts are run through **cron** at regular intervals. These scripts produce a set of standard reports, as well as a group of files containing input for these reports. In many cases, all you need to do is print the standard reports. You can automate this process by adding entries to the */usr/spool/cron/crontabs/adm* file.

Most of the accounting commands and scripts called by **runacct** and other scripts can be used in many ways to monitor system resource use. You can modify **runacct** and the other scripts to include different information, or to automatically sort the information in different ways. You can also run some of these programs manually if you occasionally need different information.

The commands described in this section enable you to create files that contain source data in predetermined formats, create new source files from data written as records to the database files, to display data written as records into those files, and to merge data from several files into a single file whose format you select.

Other commands described in this section permit you to summarize data in two or more files. You can create reports from this data, or redirect or pipe outputs from a command to files or to other commands. The following command groups are described in this section:

- Installation commands
- Disk-usage accounting commands
- Connect session accounting commands
- Process accounting commands
- The service-charge command
- Daily summary and report generating commands
- Monthly summary and report generating commands
- Printer accounting commands
- Miscellaneous accounting commands

Also described in this section are the commands that initiate accounting functions and collect data, as records, in various files. These records become the database or data source for information written into accounting reports. The accounting database files include the daily */var/adm/wtmp* login/logout file, the */var/adm/utmp* active connect-session file, the */var/adm/pacct* process accounting file, and the */var/adm/dtmp* disk usage file.

Installation Commands

There are two accounting installation commands. These commands, **/usr/bin/acct/startup** and **/usr/bin/acct/shutacct**, are scripts. These scripts must be used to ensure that the various accounting processes are enabled. The startup script is called through the **/sbin/rc2** script at system startup. The **shutacct** script is called through **/sbin/rc0** at system shutdown.

For detailed descriptions of these two commands, see the **acct** manual page.

Disk-Usage Accounting Commands

When automatic accounting is enabled, the accounting functions keep daily and monthly disk usage statistics. Weekly disk usage information is collected one day a week into the *disktacct* file in the */var/adm/acct/nite* subdirectory from the *dtmp* file in the */var/adm* subdirectory. The **diskusg** command produces the *dtmp* file whenever the **cron** daemon runs the **dodisk** script. Each time the **dodisk** script runs, the contents of the *disktacct* file are overwritten.

Daily disk usage information is collected from the *disktacct* file and written into the *daytacct* file in the */var/adm/acct/nite* subdirectory. The information in the *daytacct* file actually consists of all the records previously collected from the weekly *disktacct* file. However, when there is no *disktacct* file, which can happen during the first few days that automatic accounting functions are turned on, the *daytacct* file is not produced because there are no records to write into it.

A shell script and three commands gather information about disk usage in the system. These functions permit automatic disk-usage accounting to be initiated and also enable you to write total disk usage accounting records into files that you name. The functions reside in the */usr/sbin/acct* subdirectory:

dodisk This script gathers weekly disk usage accounting summary files, which become source data for accounting reports.

diskusg By default, the **dodisk** command calls this program, which produces an ASCII output file that provides a disk block count summary for each user owning files in a list of file systems provided as its input.

acctdusg This program produces the same ASCII disk usage summary as does the **diskusg** program, but the list of files provided as its input is derived from a **find** command. The **acctdusg** command produces a more complete total accounting file at the expense of a substantial amount of time.

acctdisk This program, which is called by **dodisk**, uses the file produced by the **acctdusg** or **diskusg** commands to produce a binary total accounting file.

Both **diskusg** and **acctdusg** produce ASCII records that contain the same information: a user name, a UID, and the number of disk blocks in files owned by the user. Both programs compare the UIDs of files to the UIDs in the password file to determine the user name and also to identify files owned by "`no one`." The programs, however, use different methods to identify the files belonging to each user and to `no one`, to allocate blocks for linked files, and to allocate blocks for empty directories.

The **diskusg** command requires a list of special file names and reads the inodes of those file systems. It finds the UID of the owner, determines the number of blocks in the file, and adds it to the total for that user. This method is fast, but it does not account for links. The blocks in a file with links are credited to the users who own the links. Files belong to `no one` if the UID is not in the password file; these files are identified by special file and inode numbers. The blocks assigned to empty directories are counted.

The **acctdusg** command requires a list of pathnames, which is usually generated by **find**. For each file in the list, **acctdusg** finds the UID of the owner. Files are assigned to `no one` if the UID is not in the password file, which are files accessible by *root* that have encrypted passwords for each user. Files that are not under the home directory of the user (as specified in the password file) are also assigned to `no one`. Files belonging to `no one` are identified by pathname. The blocks belonging to empty directories are not counted.

The **acctdusg** computes the number of blocks for each user as follows:

- If a file has one hard link (the normal case), all blocks are allocated to the owner.
- If a file has more than one hard link, the number of blocks in the file is divided by the number of links, and the result is allocated to the owner of each link.
- If a file is a symbolic link, only the space needed for the link name (usually one block) is allocated to the owner.

To modify the behavior of **dodisk**, you can change the way it is called through **cron**. Because **dodisk** is a script, you can also edit it for more extensive customization of disk accounting. You may also want to call **diskusg** or **acctdsk** occasionally to provide information that is not normally needed. The following sections provide more information about **dodisk** and the commands it calls.

The dodisk Command

Normally, the **cron** daemon runs the **dodisk** script. By default, **dodisk** calls **diskusg** for all special files listed in */etc/fstab*.

The **dodisk** syntax is as follows:

/usr/sbin/acct/dodisk [**-o**] [*files*]

With the **-o** flag, **dodisk** calls the **acctdusg** program instead of **diskusg** and gives it the list of pathnames generated by **find**.

When you do not specify the **-o** flag, the **dodisk** *files* argument must specify only the special file names of mountable file systems. When you specify both **-o** and one or more files, *files* must specify the mount points of mounted file systems.

For example, you can change the way **dodisk** behaves for automatic accounting by editing the *crontab/root* file. The following entry is typical:

```
0 3 * * 4 /usr/sbin/acct/dodisk -o /usr1 /usr2 2>/var/adm/diskdiag
```

This entry calls the **acctdusg** program for the file systems mounted on */usr1* and */usr2*.

Each time the **dodisk** script runs, it overwrites the contents of the *disktacct* file.

The diskusg Command

Normally, the **diskusg** command is called from the **dodisk** script as follows:

/usr/sbin/diskusg *filelist /var/adm/dtmp*

The *filelist* consists of the raw device names listed in */etc/fstab*. The intermediate disk accounting records written to */var/adm/dtmp* are used as input to **acctdisk**, which converts the *dtmp* information to binary total accounting records. Subsequently, these total accounting records are merged with other total accounting records to produce a daily total disk-usage report.

The **diskusg** command performs the following functions:

- Reads the inodes of the specified file systems.
- Determines the size of a file and the UID of the owner from the inode information.
- Determines the login name of the owner from the */etc/passwd* file.
- For each user listed in *passwd*, accumulates the total disk usage. When all the file systems have been processed, sorts user records by UID and writes them to the standard output.
- For each file with a UID that is not in *passwd*, writes the special file name and inode number to the standard output.

The following example is typical:

```
$ /usr/sbin/acct/diskusg
/dev/rdsk/0s0*C
0     root     63652
1     daemon      84
2     bin      71144
4     adm        976
5     uucp      3324
9     lp        1928
322   homer        2
521   whistler     2
943   cellini    363
1002  alston     212
1016  pollock      92
1098  hopper     317
```

Refer to the manpages for the **diskusg** command for mor information.

The records for `no one` consist of the special filename, the inode number, and the user ID. You can find the pathnames of these files with the **ncheck** command, which is described in the *OSF/1 Network Application Programmer's Guide*. These records identify users who are avoiding disk-usage charges or identify files belonging to users who no longer have accounts on the system.

The acctdusg Command

Normally, the **acctdusg** command is called from the **dodisk** script by using an **-o** flag. If you edit the *crontab* entry for **root**, this command can be part of automatic accounting. In the **dodisk** script, the **acctdusg** command is used in the following way:

find *dir* **-xdev -print | /usr/sbin/acct/acctdusg /var/adm/dtmp**

The directories in *dir* are supplied on the **dodisk** command line. For each file in the list, **acctdusg** identifies the owner, computes the number of blocks allocated to the file, and adds this value to the total number of disk blocks held by the user.

The computed value is found in the following way: when there is one hard link to a file (the normal case), the named user is charged for the disk space. When two or more users have links to the same file, **acctdusg** charges each user with an equal part of the total disk space occupied by the file. As an example, when a user has a link to a 1000-block file, **acctdusg** charges the owner and the user declaring the link with 500 blocks each.

The **acctdusg** command searches the */etc/passwd* file, or the alternate password file specified with the **-p** flag, for user login names and ID numbers. Whenever a match with a user login name in any file found with the find command occurs, an output record is written to the *dtmp* file. Each output record produced by this command has the same 3-column format previously described for records written by the **diskusg** command in "The diskusg Command" on page 15-24.

The following command and output are typical:

find /usr2 -xdev -print | /usr/sbin/acct/acctdusg

```
00000    root      63652
00001    daemon       84
00002    bin       71144
00004    adm         976
00005    uucp       3324
00009    lp         1928
00943    cellini     363
01002    alston      212
01016    pollock      92
01098    hoff        317
```

Two flags are associated with the **acctdusg** script. One of these flags permits you to list any pathnames for which a user has no name or user ID. The other flag permits you to name a password file other than the default password file */etc/passwd*.

-u *file* Pathnames for any files belonging to no one are written to *file*. This flag could be useful for finding users who want to avoid disk-usage charges.

-p *file* When this flag is not used, */etc/passwd* is searched for user login names and user IDs. When this flag is used, *file* is searched instead.

The acctdisk Command

The **acctdisk** command produces a complete disk accounting file. Normally, the **acctdisk** command is called from the **dodisk** script and reads the *dtmp* file produced by either **diskusg** or **acctdusg**.

The **acctdisk** command converts the *dtmp* records into a total disk accounting record and writes it to standard output. The total disk accounting records are usually merged with other accounting records using the **acctmerg** command to produce a total accounting report that includes disk-usage records. In the **dodisk** script, the **acctdisk** command is used in the following way:

```
/usr/sbin/acct/acctdisk < /var/adm/dtmp > /var/adm/acct/nite/disktmp
```

In the preceding example, the **acctdisk** command takes the file */var/adm/dtmp* as its redirected (<) standard input and writes converted binary records to the standard output. Subsequently, the temporary file *disktmp* is moved to */var/adm/acct/nite/disktacct*. It is the *disktacct* records that are merged with other records to produce a report with a selected format.

Connect-Session Accounting

The kernel automatically maintains the two system login files, */etc/utmp* and */var/adm/wtmp*. All active logins and logouts are recorded in the *utmp* file and user *login/logout* history is accumulated in the *wtmp* file. The information they provide is used to write reformatted records to various connect-session files and reports.

The record format for the various *wtmp* files (there are four of them) is established by the type *utmp* structure, which is defined in the *utmp.h* header file and in "The fwtmp Command" on page 15-29 (refer to Table 15-2 on page 15-8 for a brief description of the various *wtmp* files). Records in these files contain the following information.

- The name of the terminal and line from which the connect session took place.
- The user login name and process ID.
- Type of entry, which identifies the type of record written. For example, a login record is type 8.
- Exit status.
- Time information consisting of the time and date together with the time in seconds since the Epoch. The Epoch is referenced absolutely to 1 Jan 1970.

The *wtmp* file is a daily connect-session database that is used as a source for information written to other connect-session files or reports. As such, its records provide little directly useful information. However, commands such as **ac** and **acctcon1** convert these *wtmp* file records to useful connect-session accounting records.

There are seven accounting functions that obtain or modify information about system connect sessions. These functions, which reside in the */usr/sbin* and */usr/sbin/acct* subdirectories, consist of the following six programs and single script:

acctwtmp This program permits you to write records to the *wtmp* file from the standard input.

fwtmp This program prints records in *wtmp* formatted files to the standard output. The *wtmp* formatted files are:

- */var/adm/wtmp* The current daily connect-session database file, which is written whenever a user logs in or out of the system.
- */etc/utmp* The daily user login/logout connect session database file, which is written whenever a user calls a process that produces a connect session.
- *wtmp.mmdd* The daily previous day connect-session database. This is a copy of the previous day's */var/adm/wtmp* file.
- *ctmp* The connect-session file produced when the **acctcon1** command produces a connect-session record for each user who logged onto the system during the current day.

wtmpfix When the date command is used to set a different date on a system, a pair of date change records is written to the *wtmp* file. As a result of the date change, all records written after the changed date are corrupted. This program normalizes *wtmp* connect-session records that span the date change and also validates login names written to the user login name field in the *wtmp* file.

ac This program provides a total connect-session record for the entire system and for each user of the system.

acctcon1 This program summarizes information in the *wtmp* file and writes ASCII records to the standard output as one line for each connect session.

acctcon2 This program uses the output produced by **acctcon1** as input and writes a total connect-session accounting record to the standard output.

prctmp This script prints the contents of the session-record file */var/adm/acct/nite/ctmp* (refer to Table 15-2 on page 15-8) to the standard output.

The acctwtmp Command

The **/usr/sbin/acctwtmp** command writes a string and a date stamp to the standard output. A number of scripts, such as **startup** and **shutdown**, call this command to enter the time of an event and a description in the */var/adm/wtmp* file.

For detailed information, see the **fwtmp** manual page in the*OSF/1 Network Application Programmer's Guide*.

The fwtmp Command

The **/usr/sbin/fwtmp** command reads from the standard input and writes to the standard output, converting binary records in the */var/adm/wtmp* file to formatted ASCII records. An ASCII version of the *wtmp* file is useful for editing, repairing bad records, and maintaining the file in general.

Normally, information in record fields of the *wtmp* database file is written as binary data by the kernel or by **init** and **login** during the life of the *wtmp* file. The *wtmp* database file records have nine fields that are formatted according to members of a type *utmp* structure defined in the *utmp.h* include file. The **fwtmp** command is also capable of writing properly formatted ASCII records from standard input into a file when you use its **-i** flag. The syntax of the **fwtmp** command is as follows:

/usr/sbin/acct/fwtmp [**-ic**]

As an example, enter a command similar to the following for converting binary *wtmp* records in the type *utmp* structure format to an ASCII file written to the standard output:

```
/usr/sbin/acct/fwtmp < /var/adm/wtmp.temp
/usr/sbin/acct/fwtmp -c < /var/adm/wtmp
rm /var/adm/wtmp.temp
```

The three commands in this example are necessary because you cannot specify the *wtmp* file as both the input and output file without destroying the contents of the output file. The three commands shown above ensure that this does not happen.

The /usr/sbin/ac Command

The **ac** command reads the */var/adm/wtmp* file, calculates the total connect time for all users and for specified individual users, prints the results, and is not called by any of the automatic accounting scripts supplied with the system. If you want to generate connect-time reports automatically, you can edit the **runacct** script or add an entry to the *crontab* file for **adm**. You can also call the **ac** command manually whenever you need the information it provides.

The **ac** command has the following syntax:

/usr/sbin/ac [**-dp**] [**-w***filename*] [*users*]

By default, **ac** outputs the total connect time for all system users during the lifetime of the current *wtmp*. The units are hundredths of hours. The following example is typical:

/usr/sbin/ac
```
total 48804.26
```

The **ac** command can also output connect times for specified users and the total connect time for these users. The following example is typical:

/usr/sbin/ac danc
```
danc 12.51
total 12.52
```

For information about flags, refer to the **ac** command manpages.

The following example illustrates how the **-p** flag might be used:

/usr/sbin/ac -p
```
buckler       61.44
fujimori     530.94
newsnug      122.38
dara           0.10
dma          179.18
root         185.98
buchman      339.33
danc          12.51
russell       53.96
hoff         200.43
jrp          123.96
hermi        157.81
total       1968.02
```

Notice that a total for the users listed is the last line printed to the standard output. Connect-session outputs produced by the **ac** command are useful for performing diagnostics and when connect-session time is charged to users of systems. Permanent periodic user files can be produced by redirection.

The acctcon1 Command

When automatic accounting is enabled and the */var/adm/wtmp* file exists, and the **runacct** script is typically executed once a day. The contents of the daily *wtmp* file are copied to the */var/adm/acct/nite* subdirectory as file *wtmp.mmdd* (where *mmdd* is the month and date that the *wtmp* file records were accumulated). Subsequently, the **runacct** script processes the *wtmp.mmdd* file to obtain useful connect-session records. The **acctcon1** command produces these connect-session records.

The **/usr/sbin/acct/acctcon1** command, together with the **acctcon2** and **prctmp** commands, permit you to create useful connect-session records in files that you can use as reports. The **acctcon1** command converts login/logout records in the */var/adm/wtmp.mmdd* file to similar records, but writes a single record for each connect session.

The **runacct** script normally redirects the output of the daily *wtmp.mmdd* file as binary input to the *lineuse* and *reboots* files in the */var/adm/acct/nite* subdirectory. The syntax of the **acctcon1** command is:

/usr/sbin/acct/acctcon1 [**-l** *file*] [**-o** *file*] [**-p**] [**-t**]

Three ASCII files, *lineuse*, *reboots*, and *logsess*, are created by this command in the */var/adm* subdirectory. The *lineuse* file, which is obtained using the **-l** flag, recovers daily *wtmp* file records that list user name, the reference designation of the ports on which the user logged into the system, and the date and timestamp of the currently active connect session (see the **-l** flag in this section). The **acctcon1** command rewrites the */var/adm/wtmp* input file records as shown in the following typical line-usage file:

```
TOTAL DURATION IS 57 MINUTES
LINE      MINUTES   PERCENT      # SESS    # ON   # OFF
pty/ttyp4      37        64           3       3       7
console        26        45           2       2       4
pty/ttyp5       7        11           1       1       3
pty/ttyp6       0         0           0       0       2
TOTALS         69         -           6       6      16
```

The *reboots* file, which is obtained using the **-o** flag, recovers daily *wtmp* file records that define system reboot operations. The *reboots* file is written in an overall record format for the accounting period during which the */var/adm/wtmp* file is active (see the **-o** flag description earlier in this section). This file lists a starting time, an ending time, the number of restarts, and the number of date changes as shown in the following example *reboot* file.

```
from Fri Sep 7 07:20:12 1990 EDT
to Fri Sep 7 12:56:42 1990 EDT
2 date changes                  The number of times the date was changed.
2 acctg off                     The number of times accounting functions were turned off.
0 run-level S                   The number of times accounting functions ran in single-user mode.
2 system boot                   The number of times the system was rebooted.
2 acctg on                      The number of times accounting functions were turned on.
1 acctcon1                      The number of times the acctcon1 command was issued.
```

The comments in the third column in the preceding example do not normally appear in a typical output file; they are provided in this description to explain the purpose of the second column.

The *logsess* file, which the **acctcon1** command uses as its default output, has the following format:

```
   1          2       3        4       5          6      7  8     9        10   11
285212673     1192    hoff     85        0    616883748 Jul 19 16:35:48 1990 EST
285212673     1033    tom      10        0    616883837 Jul 19 16:37:17 1990 EST
285212673        0    root   1345     2852    616883855 Jul 19 16:37:35 1990 EST
285212673     1120    jim       0       62    616888058 Jul 19 17:47:38 1990 EST
```

The numbers in boldface type are not part of the output, but are shown for reference. The default format output columns have the following significance:

1 The device address expressed as a decimal equivalent of the major/minor device address at which the connection was activated.

2 The user ID assigned for the connect session record.

3 The user login name under which the session took place.

4 The total number of prime-time seconds for the connect session.

5 The total number of non-prime-time seconds for the connect session.

6 The number of seconds since the Epoch. The Epoch is referenced absolutely to 0 hours, 0 minutes 0 seconds, 1 Jan 1970 GMT.

7 The month of the year expressed as an initial-capitalized three-letter string.

8 The day of the month expressed as a decimal number.

9 The connect-session starting time expressed in hours, minutes, and seconds.

10 The year expressed as a four digit number.

11 The name of the current timezone.

For all dates and timestamps, the order of the information depends on the locale. All the examples in this section show the default order. The order can be changed with the NLTIME environment variable. See the **en(4)** manpage for more information about the locale.

The prctmp Command

The **/usr/sbin/acct/prctmp** command writes headings on the output files created by the **/usr/sbin/acct/acctcon1** command. Its syntax is as follows:

prctmp [*file*]

You can use the **prctmp** command to output the connect-session record file, which is specified by the *file* argument and created by the **acctcon1** command (this is normally the */var/adm/acct/nite/ctmp* file). The */var/adm/acct/nite/ctmp* file is produced when the **runacct** script is processed. The **prctmp** command is used in the following way:

/usr/sbin/acct/acctcon1 < /var/adm/wtmp | /usr/sbin/acct/prctmp > logsess.hdg

The **acctcon1** command produces a connect-session output file using the */var/adm/wtmp* database file as the redirected input. The output of **accrcon1** is piped as input to the **prctmp** command. The output produced by this command, which is normally printed to the standard output, is redirected to the *logsess.hdg* file. The following example shows typical output:

```
Sep 07 17:50 1992 SESSIONS, SORTED BY ENDING TIME Page 1
MAJ/MIN                  CONNECT SECS    START TIME    SESS        START
DEVICE      UID   LOGIN   PRM     NPRM      (NUM)       DATE        TIME
        0     0   root    741        0     652713628    Fri Sep 7   09:20:28 1990 EST
285212676  1114   josh    375        0     652716408    Fri Sep 7   10:06:48 1990 EST
285212676  1048   hal      78        0     652716791    Fri Sep 7   10:13:11 1990 EST
285212677  1238   robertc 392        0     652716599    Fri Sep 7   10:09:59 1990 EST
        0  9261   hoff    253     2808     652715761    Fri Sep 7   09:56:01 1990 EST
```

The column headings in this output example are placed at the beginning of a connect-session file such as the */var/adm/acct/nite/ctmp* file that is described in the previous section. These column headings are not normally written to the connect-session files by any of the scripts that are executed during automatic accounting. However, with the addition of column headings, the connect-session files become reports.

The acctcon2 Command

The **/usr/sbin/acct/acctcon2** command is usually called by the **runacct** script to convert ASCII connect session records (previously produced by the **acctcon1** command) into complete binary connect-session accounting records. The binary total accounting records produced by this command are often merged with other total accounting records using the **acctmerg** command to produce daily reports.

For example, the */var/adm/acct/nite/ctmp* file that is produced when the **runacct** script calls the **acctcon1** command becomes the input to the **acctcon2** command. The **runacct** script uses the **acctcon2** command in conjunction with the **acctmerg** command in the following way to produce a temporary connect-session total accounting file:

/usr/sbin/acct/acctcon2 > /var/adm/logacct.tmp

In this example the ASCII *ctmp* file, which is produced by the **acctcon1** command, is the redirected input to the **acctcon2** command. It produces the binary total accounting file named */var/adm/logacct.tmp*. Because this file is not human-readable, it normally serves as input to the **acctmerg** command, which produces total accounting output files that combine both connect-session and process-accounting information in their records.

Process Accounting

The process accounting database file */var/adm/pacct* is maintained by the kernel. Whenever a user enters a command, or whenever a command is executed from a script, statistics about the resource usage that is required to execute the command are written to the daily *pacct* file. When the **cron** daemon runs the **runacct** script each day, records in the *pacct* file are stored in other *pacct* files in the */var/adm/acct/nite* subdirectory. This section describes the commands that produce these files in the */var/adm/acct/nite* subdirectory.

The record format for the *pacct* files is established by the type *acct* structure, which is defined in the */usr/include/sys/acct.h* header file (refer to Table 15-2 on page 15-8 for a brief description of the *pacct* files in the */var/adm* and the */var/adm/acct/nite* subdirectories). Records in the *pacct* files contain the following information:

- Each command name executed for the period during which the */var/adm/pacct* file was active
- The user and group ID
- The terminal reference designation (tty name) through which the command was issued
- The time (start) at which the command was executed
- The amount of real time (CPU plus system time) required to execute the command
- The amount of CPU time required
- The mean amount of memory required to execute the command, in Kilobytes
- The number of characters transferred by the process
- The number of blocks read and written
- The accounting flag
- The process exit status

Process accounting commands permit you to compile statistics about commands and the resources used to process them. These command execution statistics tell you who invoked the command, the amount of memory used, the total machine time needed to execute them, the CPU time used, the terminal from which any command was invoked, and the amount of memory space needed for the command. To obtain process accounting statistics you must do the following:

1. Turn process accounting on.
2. Check that process accounting file /var/adm/pacct is not excessively large.
3. Generate process accounting records, files, and reports.
4. Generate command summary reports.
5. Create periodic total accounting records, files, and reports.

Most of these tasks are automatically performed by the **runacct** script when it is invoked by the **cron** daemon. Normally, the **startup** command called through **rc2** automatically turns process accounting on. However, you can turn process accounting on and off by entering the appropriate command.

When you have properly set up the required files for automatic accounting, automatic process accounting is enabled. When you have not enabled automatic process accounting refer to the description in "Setting Up Accounting Files" on page 15-14 for process accounting commands and what they do.

The *turnacct on* Command

Normally, the **startup** script calls the **/usr/sbin/acct/turnacct on** script. The **turnacct on** script can be invoked from the command line only when you have superuser or **adm** authority. When you issue this command, the **turnacct on** script does the following:

1. Checks for the presence of the */var/adm/pacct* process accounting file.
2. When the *pacct* file is not present, one is created.
3. When the *pacct* file exists, process accounting is turned on with the **accton** command.

The accton Command

You can turn accounting on, create your own process accounting file, and make it the active one by writing a filename other than *pacct*. However, the new file must currently exist. When you use the **/usr/sbin/acct/accton** command with a filename argument, automatic process accounting is turned on and the file named in the argument becomes the file to which the kernel writes all process accounting records. For example, when you enter the following, the kernel writes all process accounting records to the file you name using the *filename* parameter:

accton *filename*

When you use a name other than *pacct* for process accounting records, you must be sure that other process accounting files reflect the name change. These process accounting files are also used in the **ckpacct**, **runacct**, and **turnacct** scripts.

The ckpacct Command

On large multi-user systems a great number of commands can execute in a short period of time. Because of this, the */var/adm/pacct* file can become excessively large. The system accounting functions use the **/usr/sbin/acct/ckpacct** script to ensure that the *pacct* files remain of usable size. There are two scripts that permit you to control the size of the *pacct* files: **ckpacct** and **turnacct switch**. These scripts do not work when your process accounting files do not have the default process accounting file named *pacct*.

The **ckpacct** command checks the size of the *pacct* files. This command is invoked by the **cron** daemon once an hour to ensure that the *pacct* process accounting file does not become too large. The syntax of the **ckpacct** command is as follows:

ckpacct [*blocksize*]

The **ckpacct** command is called through **cron** in the following way:

```
5 * * * * /usr/sbin/acct/ckpacct
```

The **cron** daemon runs the **ckpacct** script 5 minutes after each hour to ensure that the size of the latest *pacct* file does not exceed the default block-size values. The reason this command is not executed on the hour is so that its process does not collide with any process previously invoked on the hour.

The **ckpacct** command first checks that more than 500 blocks of disk space remain available to */var/adm* subdirectory. Should less than 500 blocks of disk space be available at the time the **cron** daemon invokes the **ckpacct** command, process accounting is turned off using the **turnacct off** script and a message similar to the following is mailed to the user *adm*:

```
ckpacct: /var/adm too low on space (300 blocks) turning acctg off
```

This example shows 300, but the number of blocks can range between 0 and 499. After an hour has elapsed, the **cron** daemon runs the **ckpacct** script again. When available disk space remains less than 500 blocks, a message similar to the following is mailed to the user *adm*:

```
ckpacct:/var/adm still too low on space (310 blocks) acctg still off
```

When a sufficient amount of disk space is returned and more than 500 blocks are available in the */var/adm* subdirectory, the following message is mailed to the user *adm*:

```
ckpacct: /var/adm free space restored; turning acctg on
```

When the */var/adm* subdirectory has more than 500 blocks of free space available and the length of the */var/adm/pacct* file is less than 1000 blocks, the **ckpacct** script exits. When the */var/adm* subdirectory has more than 500 blocks of free space available and the length of the */var/adm/pacct* file is greater than 1000 blocks, a new *pacct* file is created using the **turnacct switch** command.

The *turnacct switch* Command

The **/usr/sbin/acct/turnacct switch** command permits you to create a new *pacct* file whenever the size of the current one becomes excessively large. Normally, **cron** invokes the **ckpacct** script once every hour to ensure that the size of the current *pacct* file does not exceed 1000 blocks. However, when the size of *pacct* is greater than 1000 blocks, the **turnacct switch** script is called by the **ckpacct** script in the following way:

turnacct switch

When the **turnacct switch** script is invoked, the following actions take place:

1. Process accounting is turned off.
2. The active *pacct* default file is renamed *pacct*_n_, where _n_ is an integer 1 greater than the _n_ of the active *pacct*_n_.
3. After the active *pacct* file is renamed, a new default *pacct* file is created.
4. Process accounting is restarted and whenever a command is processed, after the command has successfully executed, the kernel writes a record to the new *pacct* file.

Service Charges

The accounting system monitors "normal" resource allocation, such as CPU time and disk usage. When a site wishes to keep track of other services, the **/usr/sbin/acct/chargefee** script can be used. The meaning of the units charged to the user by this script is determined by the site.

Scaling Charge Units

Because you can only charge users units in integer quantities, you should have a scale that establishes the lowest charge unit as the smallest charge quantum. As an example, suppose that the following services are charged to users:

- File backup to disk
- File recovery from disk
- File backup to tape
- File recovery from tape
- Software technical assistance by phone
- Software technical assistance in person

Obviously, more work must be done by service personnel to mount and dismount tapes from a tape drive than to issue disk backup commands from a terminal, and far more must be done when software technical assistance is rendered in person. Assuming you do not have to change disk cartridges your smallest unit charge is probably for disk backup and recovery. Consequently, you might charge 1 unit each for 1 and 2 in the above list, 5 units for 3 and 4, 5 units per minute for 5, and 7 units per minute for 6.

Charging Unit Fees

The **chargefee** command permits you to pass charges onto users who avail themselves of your system services. When you enter the **chargefee** command, a *fee* file is created in the */var/adm* subdirectory and a single *chargefee* record written to it. The syntax of the **chargefee** command is:

chargefee *login_name units*

For example, to charge 72 units to the user whose name is *josh*, enter the following command:

chargefee josh 72

When the command in this example is executed, the following record is written to the *fee* file in the */var/adm* subdirectory:

```
1114 josh 0 0 0 0 0 0 0 0 0 0 72
```

Records in the *fee* file have 13 fields. However, information is only written in the first, second, and last fields. The first field contains the user ID from the */etc/passwd* file. The second field contains the username, and the last field contains the value you wrote as the number of units to charge in the **chargefee** command. Unused fields contain 0s (zeros).

Each time you subsequently charge other positive or negative units to the same user, a new record is added to the */var/adm/fee* file. Negative units are preceded with a - (dash). When summary records that use the information in the *fee* file are produced, the total number of units charged to each user is computed from the records in this file.

Daily Files

The daily, summary, and total files are produced when the **cron** daemon runs the **runacct** script. These daily files are stored in the */var/adm*, */var/adm/acct/nite*, and *var/adm/acct/sum* subdirectories. With the exception of the database files, which are located in the */var/adm* directory, most of the files that are created and maintained on a daily basis are stored in the */var/adm/acct/nite*, and *var/adm/acct/sum* subdirectories (refer to Table 15-2 on page 15-8).

The Daily File Directory

In addition to the database files normally stored in the */var/adm* subdirectory, this subdirectory stores three files that are produced on a daily basis by the **runacct** script. The daily database files are:

lineuse — This file is produced by the **runacct** script to record information about terminal line use.

reboots — This file is produced by the **runacct** script to record the number of times the system was rebooted during the past 24-hour period.

logsess — This file is produced to record the number of users who have logged in or out of the system during the past 24-hour period.

Daily /var/adm/acct/sum Files

Many daily summary files are produced and processed by the **runacct** command. The number of files stored in the */var/adm/acct/sum* subdirectories can vary from just a few to dozens, depending on the time of the month it is and what system conditions exist when the **runacct** script executes. The daily summary files are:

daycms — The **runacct** script calls the **acctcms** command to produce the daily binary *daycms* command summary file. When **runacct** is processed, the number of times each command listed in the daily */var/adm/pacct* file was invoked is used to compile a statistical summary record. A single summary record for each command used during the 24-hour period covered by the **cron** daemon is written to this *daycms* file.

cmsprev Before the daily *daycms* file is created, the **runacct** script copies the existing *daycms* summary file as the *cmsprev* binary file. The next time (following day) the **runacct** script runs, command summary records in the *cmsprev* and the new *daycms* files are summed to produce the new daily */var/adm/acct/sum/cms* file.

cms The **runacct** script also calls the **acctcms** command to produce this third total command summary binary file. To obtain the records in this file, the **acctcms** command sums the number of times each command listed in the *daycms* file occurred with the number of times the same command occurred in the daily *cmsprev* file. A single summary record for each command found in either file is written to the *cms* file. The *cms* file accumulates command summary records every day until it is truncated or deleted, either from the command line, or from a script.

tacct.mmdd When accounting is automated and the **cron** daemon is run, the **runacct** script produces a *daytacct* file and stores it in the */var/adm/acct/nite* subdirectory. A copy of the daily *daytacct* file is written to the */var/adm/acct/sum* subdirectory as the *tacct.mmdd* binary file. This binary file provides the daily and monthly total accounting summary and report files.

tacct When the **runacct** script is run by the **cron** daemon, a new *tacct* file is created in the */var/adm/sum* subdirectory each day. The records in the current */sum/tacctprev* and the current */sum/tacct.mmdd* files are written to the new */sum/tacct* file. The records contained in the latest *tacct.mmdd* file are obtained from the daily */var/adm/acct/nite/daytacct* file. By summing records in the *tacctprev* and current day *tacct* files, a continuous total accounting record file is built. On the first day of each month, the *tacct* file contains the total accounting records for the entire previous month.

tacctprev After the **runacct** script produces the total account summary records in the *tacct* file, the current *tacct* file records are copied to the *tacctprev* file. In this way, the *tacctprev* file records and current daily *tacct* records are combined to create each new daily *tacct* file.

loginlog When **cron** calls **runacct**, it in turn calls the **lastlogin** script, which creates an ASCII file called *loginlog*. Refer to "The lastlogin Command" on page 15-59 for the file format.

ptacct.mmdd This daily binary file is a summary of records collected from all the *pacct* files in the */var/adm* subdirectory for the current 24-hour accounting period. The **runacct** script produces *ptacct.mmdd* files when it is called from the **cron** daemon each accounting day.

rpt.mmdd Each day the **runacct** script creates a report file when it calls the **prdaily** script, which produces the various report records written to these daily files. Each daily report file is dated and provides different categories of information.

The daily report consists of the following:

1. A line-usage summary and an overall record report produced by the **acctcon** command
2. A daily total accounting report produced by the **acctmerg** command
3. A daily command summary produced by the **acctcms** command
4. A total monthly command summary for the entire month up to the current date, which is also produced by the **acctcms** command.

The *rpt.mmdd* report contains this information in the following formats:

- The initial page specifies a daily report title for a named system and for a specified accounting period. This page provides a printout of the daily *reboot* and *lineuse* files in the */var/adm* subdirectory. These line use records provide you with information about the number of times the system was booted, the number of times accounting was turned on, shutdowns, power-fail recoveries, and any other information that gets written into the */var/adm/wtmp* file during the previous day.
- The second group of records written to the initial page provides daily *lineuse* file information, which includes: the total number of minutes any terminal was connected to the system, the total number of minutes the line was used during the accounting period, the total number of minutes the connection was in use divided by the time the line was in use, and the number of times the port was accessed for a connect session.
- The daily total usage report page for the system, which the **runacct** script produces when it calls the **prdaily** script, is printed from the total accounting records stored in the *daytacct* file in the */va/adm/acct/sum* subdirectory. The output format of this part of the daily report consists of 18 columns of daily total account information (refer to the **acctmerg** command in "The acctmerg Command" on page 15-53 for a description of the output format).
- The daily command summary report page for the system, which the **runacct** script produces when it calls the **prdaily** script, is printed from the daily summary accounting records stored in the *daycms* file in the */var/adm/acct/sum* subdirectory. The output format of this part of the daily report consists of 10 columns of daily command summary information (refer to the **acctcms** command for a description of the output format).

- The monthly command summary report page for the system, which the **runacct** script produces when it calls the **prdaily** script, is printed from all the daily summary accounting records stored in the *cms* file in the */var/adm/acct/sum* subdirectory. This information consists of a summary of all commands used during the month up to the day before the indicated date. The output format of this part of the daily report is the same as that obtained for the daily command summary.

Daily /var/adm/acct/nite Files

The **runacct** command produces and processes many daily files during nonprime-time nighttime hours. The number of files stored in the */var/adm/acct/nite* subdirectories can vary from just a few to several, depending on the time of the month and what system conditions exist when the **runacct** script executes. The daily nonprime-time summary files are as follows:

active During execution, the **runacct** script writes messages to the *active* file to indicate completed tasks. The following example is typical:

```
file setups complete
wtmp processing complete
connect acctg complete
process acctg complete for /var/adm/Spacct1.0914
all process actg complete for 0914
tacct merge to create daytacct complete
no fees
merged disk records
updated sum/tacct
command summaries complete
```

Whenever the **runacct** script detects an error, it produces an alternate *active.mmdd* ASCII file (where *mmdd* is the month and date the file was created).

daycms The **runacct** script calls the **acctcms** command to produce this *daycms* command summary, which is an ASCII copy of the */var/adm/acct/sum/daycms* binary file. This file allows any user to display or print a daily command summary.

cms The **runacct** script also calls the **acctcms** command to produce this second total command summary, which is an ASCII copy of the */var/adm/acct/sum/cms* binary file. As with the */var/adm/acct/nite/daycms* file, any user can display or print a daily total command summary.

daytacct The *daytacct* binary file is the total accounting file, which combines both the *ctacct.mmdd* daily connect-session records and the daily *ptacct.mmdd* process accounting records. The **runacct** script produces the *daytacct* file.

wtmp.mmdd The **runacct** script produces the *wtmp.mmdd* file. This file is a dated binary copy of the daily */var/adm/wtmp* file.

ctmp A new *ctmp* file is produced each day when the **runacct** script calls the **acctcon1** command. The *ctmp* file records are obtained from the */var/adm/wtmp* file after it has been processed by the **wtmpfix** command. The *ctmp* file records are also written to the temporary *ctacct.mmdd* file.

owtmp The *owtmp* file is a collection of all the */var/adm/wtmp* connect-session binary records for the current day. The **runacct** script writes the records in the binary daily *wtmp* file to this file.

Daily and Summary File Generating Commands

All but two of the daily and summary file generating commands are called from the **runacct** script and produce the files that are stored in the */var/adm/acct/nite* and */var/adm/acct/sum* subdirectories. The exceptions to this are the **acctcom** and **sa** commands, which permit all system users to obtain accounting information in selected output formats. The **runacct** script is also described in this section.

This section describes 10 daily and summary file generating commands, most of which are called from the **runacct** script and some of the other scripts. These commands provide information about connect-sessions, disk usage, and processes spawned during system operation in the 24-hour period covered by running of the **cron** daemon.

Some of these daily file generating commands are also used to produce monthly summary files, which are described in "Monthly Summary and Report Generating Commands" on page 15-59. The commands and scripts described in this section may also be entered on the command line so that the information in the files that these commands produce can be either printed or viewed from the terminal display. These commands include the following six programs and four scripts:

runacct A script that produces the daily and summary files that are stored in the */var/adm/acct/nite* and */var/adm/acct/sum* subdirectories. This script is usually run when the **cron** daemon calls it during late nonprime-time hours, when few user are expected to be logged onto the system. However, **runacct** may also be called from an active command line and it can be started in any of its 13 states.

acctcom Reads process accounting records from files that you specify as arguments with the command, from standard input or from the currently active */var/adm/pacct* file. The use of the **acctcom** command is not restricted to superuser or administrative authority. Many options with this command permit you to specify the output format.

acctcms Called by the **runacct** script to read process accounting records from the database files that are specified as arguments. It combines and sorts their records for identically named commands and writes their statistics in binary format to the standard output. Options permit variations in the output sort and its format.

acctmerg Called by the **runacct** script. It combines process connect-time, fee, disk-usage, and printer queueing total accounting records, puts them into an established output format, and sends the resulting report to the standard output. Total accounting records are read from the standard input along with as many as nine files that you name. The accounting file produced as output can have as many as 18 columns of accounting information.

acctprc1 Called by the **runacct** script. It reads records from standard input in the established accounting format and adds usernames to each process. User names are obtained from a file that you specify as an argument or from the default */etc/passwd* file.

acctprc2 Called by the **runacct** script. It reads the records written by the **acctprc1** command as input and summarizes them by user ID and name. These records, sorted by user name, are written to standard output as total accounting records in the total accounting format.

prtacct Called by the **runacct** script. It formats and displays any total accounting file specified as an argument. The format of files specified as input must be in the established accounting format. This command keys your output file to connect time, to process time, to disk usage, or to printer usage. You may also specify a title for the output file.

prdaily Called by the **runacct** script. It formats an ASCII output accounting file produced on a previous day. Records written to this file are contained in the */var/adm/acct/sum/rpt.mmdd* files.

lastlogin A script called by the **runacct** script that updates the */var/adm/acct/sum/loginglog* file to show the last date each user logged onto the system.

printpw Reads the *passwd* data base and compiles a list of the last date that system users logged in. This information is used to age user accounts. You should set a period of time that limits how long a user account can remain inactive.

Normally, all user files are backed up, but because there never seems to be enough disk space, you should set a policy for the use of this valuable resource.

The runacct Command

The **runacct** script produces the daily and summary accounting files. This script calls most of the scripts and commands used by the system automatic accounting function. The daily and summary files produced by **runacct** are described in "Daily Files" on page 15-39. The **runacct** script processes connect-time, fee, disk usage, and process accounting database files to create the daily and summary files stored in the */var/adm/acct/nite* and */var/adm/acct/sum* subdirectories.

The **cron** daemon calls the **runacct** script daily during non-prime time when most users are not expected be logged into the system. Most often the hour chosen is 2:00a.m. You can also invoke the **runacct** script from the command line. Running this script from the command line is not recommended unless the automatic version did not complete.

The **runacct** script is designed to be self-preserving in the sense that whenever an error occurs **runacct** takes care to let you know what happened and to preserve any files it has previously produced. The protection features of **runacct** are:

- Provides diagnostic file */var/adm/acct/nite/accterr* to tell you what errors were encountered.
- Provides an account of the operating states it has completed and what it has done (refer to the description of the */var/adm/acct/nite/active* file in "Daily /var/adm/acct/nite Files" on page 15-42).
- Uses temporary lock files to ensure that it is not invoked more than once at the same time.
- Writes the last date it was executed by the **cron** daemon in the */var/adm/acct/nite/lastdate* file.
- When an error is detected, writes a message to the console and sends mail to **root** and **adm**, removes the locks, saves the diagnostic files and error messages, and halts.

Reentrant States

The **runacct** script is operated as a finite 13-state machine. Should **runacct** stall or exit before completion, you can restart it from any of its states. The syntax of **runacct** is:

/usr/sbin/acct/runacct [*mmdd*] [*state*]

Whenever it is called, the **runacct** script is processed in 13 separate restartable states. When the **runacct** process completes each state, the name of the next state to undergo execution is written to the *statefile* file in the */var/adm/acct/nite* subdirectory. The **runacct** script processes the various states named in the second column of the following table in the order listed in the first column of Table 15-3.

When **runacct** executes, each of its reentrant processes is controlled by the information contained in the *statefile* file. Each time one of its reentrant states is completed the statename is entered in the *statefile* file. After completion of its current state, **runacct** examines the *statefile* file to determine what state to enter next. You can also restart **runacct** manually. Normally, you do this to recover from a **runacct** failure, or to recreate lost or corrupted files.

Failure Recovery

When the **runacct** script is called with no arguments, as it is from the */usr/spool/cron/crontabs/adm* file, this process assumes that this is the first call to **runacct**, and normal lock protection is used.

When **runacct** fails or terminates before completion, it must be restarted from its last known state, which is contained in the */var/adm/acct/statefile*. To restart **runacct**, specify the day and month for which accounting files must be obtained as follows:

/usr/sbin/acct/runacct [*mmdd*] [*state*]

In the previous command syntax, *mmdd* is the month of the year and day of the month expressed as 2-digit integers. When the */var/adm/acct/statefile* is not present, or you wish to start **runacct** from some other state, you must specify it as the *state* parameter (refer to Table 15-3). When **runacct** fails or terminates before completion, you should check for the following possible events:

- A previous system crash
- Insufficient disk space in the */var/adm* directory
- A corrupted */var/adm/wtmp* file

To restart **runacct** from the first previous state it did not complete, perform the following steps:

1. Check whether the */var/adm/acct/nite/active.mmdd* file contains any error messages.
2. When the */var/adm/acct/nite/active* and *lock* files exist, check the */var/adm/acct/nite/accterr* file for diagnostic messages. For each of the diagnostic messages listed below, take the indicated corrective action.

   A. `ERROR: locks found. run aborted`

      One or more lock file exists. Remove the lock files and then restart runacct from its last indicated state, which is stored in the /var/adm/acct/statefile.

Table 15-3. runacct States (1 of 2)

State	Name	Description
1	SETUP	Moves the active accounting files to working files and restarts the active files by calling the **turnacct switch** command. Any process accounting file with the name *pacctn* is renamed *Spacct.mmdd* so that after **runacct** completes, only the single *pacct* default file is active. The current active */var/adm/wtmp* database file is renamed *wtmp.mmdd* and moved to the */var/adm/acct/nite* subdirectory.
2	WTMPFIX	Some date changes can cause the **acctcom** command to fail when it is called from **runacct**. This state uses the **wtmpfix** command to repair corrupted date changes when necessary and verifies the integrity of the *wtmp.mmdd* file (refer to the **wtmpfix** command description) in the */var/adm/acct/nite* subdirectory. The repaired version of the *wtmp.mmdd* file is written as the *tmpwtmp* file in the same subdirectory.
3	CONNECT1	Calls the **acctcon1** command to write connect-session records to the */var/adm/acct/nite/ctmp* file. The daily */var/adm/lineuse* and *reboots* files are also created.
4	CONNECT2	Converts connect-session records from the */var/adm/acct/nite/ctmp* file to total accounting records in the total accounting format defined by *tacct* structure members in the *tacct.h* header file.
5	PROCESS	The **acctprc1** and **acctprc2** commands are called to convert any process accounting file named *Spacct.mmdd* in the */var/adm* directory to total accounting records in the *ptacctn.mmdd* daily files in the */var/adm/acct/sum* subdirectory. The *Spacct.mmdd* and *ptacctn* files are correlated by number so that should **runacct** fail for any reason, the *Spacct.mmdd* files already processed need not be processed again.
6	MERGE	Merges the connect and process total accounting records into *daytacct* records. The *daytacct* file is stored in the */var/adm/acct/nite* subdirectory.
7	FEES	Merges records in the */var/adm/fee* file previously obtained with the **chargefee** command into total accounting records (see CONNECT2 above) in the *daytacct* file.
8	DISK	On the day following the last previous call of the **dodisk** script by the **cron** daemon this **runacct** process state merges disk accounting records with *connect*, *process*, and *fee* total accounting records in the */var/adm/acct/nite* subdirectory.

Table 15-3. runacct States (2 of 2)

State	Name	Description
9	QUEUEACCT	Sorts queue (printer) accounting records, converts them into total accounting records (see CONNECT2 above), and merges them with other total accounting records. These accounting records are not the same as those obtained with the **pac** command.
10	MERGETACCT	Merges the daily total accounting records in the */var/adm/acct/nite/daytacct* file with summary total accounting records in the */var/adm/acct/sum/tacct.mmdd* file. This is done so that the */var/adm/acct/sum/tacct* file records can be reconstructed should the files become corrupted or lost.
11	CMS	Copies the current *cms* file records to the *cmsprev* file in the */var/adm/acct/sum* subdirectory. Also merges the current *pacct* accounting records into *daycms* file records stored in the */var/adm/acct/sum* subdirectory and creates the *cms* and *daycms* ASCII files in the */var/adm/acct/nite* subdirectory.
12	USEREXIT	When the script */var/adm/siteacct* exists, and the **runacct** script enters this state, the */var/adm/siteacct* script is called to perform site-dependent accounting record processing. Any site-dependent routines or scripts that you have written should be called from this **runacct** state. For example, if you have a script that produces daily billing for resource use by individual system users, enter the command that calls this script into the script */var/adm/siteacct* and it will be executed in this state.
13	CLEANUP	Deletes all temporary files, calls the **prdaily** command to create the *rprt.mmdd* file records in the */var/adm/acct/sum* subdirectory, deletes the temporary lock files in the */var/adm/acct/nite* subdirectory, and then exits.

B. `ERROR: acctg already run for` *mmdd*`: check/var/adm/acct/nite/lastdate`

   The current month and day indicated by the content of the *lastdate* file are the same. Remove the *lastdate* file and then restart **runacct** from its last indicated state, which is stored in the */var/adm/acct/statefile*.

C. `ERROR: runacct called with invalid arguments`

D. `ERROR: turnacct switch returned rc=?`

E. `ERROR: Spacct?.mmdd*C already exists: run setup manually`

F. `ERROR: wtmpfix errors see /var/adm/acct/nite/wtmperror`

When it was called by **runacct** during the WTMPFIX state, an irreparable */var/adm/wtmp* file was found.

G. `ERROR: invalid state, check /usr/var/adm/nite/active`

During processing, **runacct** probably detected a corrupted active file. Check both the *active* and *statefile* files.

Restarting runacct

Whenever you restart **runacct**, you should always check the */var/adm/acct/nite/active* file for messages and the *acctерr* files for bad data. Also check for corrupt */var/adm/pacct* or */var/adm/wtmp* files and fix them when they are corrupted. Whenever you plan on restarting **runacct**, be sure the *lock* files have been deleted. Also, it is a good idea to delete the */var/adm/acct/nite/lastdate* file.

When **runacct** has been invoked after a failure, the */var/adm/acct/nite/statefile* information is probably wrong. Check any */var/adm/acct/nite/active* or *active.mmdd* files and restart **runacct** from its last known state.

When **runacct** has failed in its PROCESS state, remove the *pacct* or last *pacctn* file because it can be incomplete.

When **runacct** fails on more than one day successively, do the SETUP process manually by entering **runacct** script commands for its STARTUP state here.

The following examples show how the **runacct** command might be used. (These examples use the I/O redirection syntax of the Bourne shell (**sh**) and Korn shell (**ksh**). If you use the C shell (**csh**), they will not work without modification.)

```
# nohup /usr/sbin/acct/runacct 2 > /var/adm/acct/nite/accterr&
```

In the previous example, **runacct** is invoked without arguments and runs in the background. Any error messages generated during its execution are written to the *acctерr* file. When you use **nohup** with the command, you specify that signals are to be ignored and that hangups, logouts (continues to run after you logout), and quits are also to be disregarded. For example:

```
# nohup /usr/sbin/acct/runacct 0926 2 > /var/adm/nite/accterr&
```

In the previous example, **runacct** is invoked with the argument **0926**, which tells the process that only the database for 26 September should be accessed. Error messages are written as previously described.

```
# nohup /usr/sbin/acct/runacct 0926 MERGE 2 > /var/adm/nite/accterr&
```

In the previous example, **runacct** is invoked beginning with its **MERGE** state and only the database for 26 September should be accessed. Error messages are written as previously described.

The acctcom Command

The **acctcom** command prints records from any process accounting database file. This command, which is one of the few accounting commands not called from the **runacct** script, allows you or any user to view process accounting data in a wide variety of formats. You can specify how you want to view the data, either from a display terminal or as hard copy from a printer. The syntax of this command is:

/usr/bin/acctcom [*flags*] [*files*]

When you specify one or more file(s), the **acctcom** process reads each file chronologically in time-descending order according to process completion time. When you do not specify files, records in the */var/adm/pacct* file are the default ones that **acctcom** writes to the standard output. Because the **ckpacct** script keeps the *pacct* file from growing too large, a busy system can have several *pacct* files. The pathnames for all but the current file are */var/adm/pacctn*, where *n* is a unique integer assigned to additional */var/adm/pacct* files in the order they are created.

Each output record represents execution times for one completed process. The default output format includes the command name, username, tty name, process start time, process end time, real seconds, CPU seconds, and mean memory size (in kilobytes). The process summary output has the following default column-heading format:

```
ACCOUNTING RECORDS FROM: day mon date hh:mm:ss yy
COMMAND                    START     END      REAL      CPU      MEAN
NAME    USER    TTYNAME    TIME    TIME    (SECS)    (SECS)    SIZE(K)
```

The date and timestamp format is as follows:

day mon date hh:mm:ss yy

day is the day of the week, *mon* is the month, *hh:mm:ss* is the time expressed in hours (in 24-hour clock time), minutes, and seconds, and *yy* is the year expressed as four digits.

By using appropriate flags, you can also output the state of the **fork/exec** flag, `F`; the system exit value, `STAT`; the ratio of total CPU time to elapsed time, `HOG FACTOR`; the product of memory used and elapsed time, `KCORE MIN`; the ratio of user time to total (system plus user) time, `CPU FACTOR`; the number of characters transferred during I/O operations, `CHARSTRNSFD`; and the total number of blocks read or written, `BLOCKS READ`.

Whenever a process is run under root authority, the record for the command name is prefixed with a number sign (#). When a process is not assigned to a known workstation (for example, when the cron daemon runs the process), a question mark (?) is written in the `TTYNAME` column.

This command has 23 flags that select particular records from the *acct* formatted records in the *pacct* files. See the manpages for the **acctcom** command.

The acctcms Command

The **acctcms** command produces daily total summary information about all the commands used during the daily accounting period. The **acctcms** process searches specified input files for recognizable commands.

For each command found, *pacct* file records are scanned for statistics, which are then combined into a single record for each different command used during the daily accounting period. Normally, this command is called from the **runacct** script during its CMS state. However, you can use this command from the command line to produce a command summary called `Total Command Summary`, which is the title of the columnar information in the file. The syntax for **acctcms** is:

/usr/sbin/acct/acctcms [**-acjnspot**] *file* ...

This command reads each file specified by the *file* parameter, combines and sorts all records for identically named processes, and writes them in a binary format to the standard output. Files are usually organized in the *acct* record format. When you use the **-o** and **-p** flags together, the **acctcms** command produces a summary report that combines commands processed during both prime and nonprime time.

All the output summaries specify total usage (except for the number of times run), CPU minutes, and real minutes, which are split into prime and nonprime minutes. The **acctcms** command normally sorts its output in descending order according to total *kcore minutes*. The unit *kcore minutes* is a measure of the amount of storage used (in K-bytes) multiplied by the amount of time the buffer was in use. The *hog factor* is the total CPU time divided by the total real time. The default command summary output format has the following headings.

```
                                TOTAL COMMAND SUMMARY
COMMAND   NUMBER  TOTAL     TOTAL     TOTAL     MEAN     MEAN     HOG     CHAR     BLOCKS
NAME      CMDS    KOREMIN   CPU-MIN   REAL-MIN  SIZE-K   CPU-MIN FACTOR  TRNSFD   READ
```

Flags

This command has 8 flags that select the format of the total summary output file. The files produced when the **runacct** script calls **acctcms** use the **-a** and **-s** flags. The files include */var/adm/acct/sum/daycms*, */var/adm/acct/sum/cms*, */var/adm/acct/nite/daycms* and */var/adm/acct/nite/cms*. See the **acctcms** command manpages for flag information.

The following example produces an ASCII command summary file for the current daily accounting period from the default *pacct* files:

/usr/sbin/acct/acctcms -a /var/adm/pacct

Normally, the **acctcms** command is called by the **runacct** script during its `CMS` state when it is executed by the **cron** daemon. The **acctcms** command expects *pacct* is a binary file that it must convert to ASCII output.

In the ASCII total summary file, each record contains more than 80 characters. Consequently, more than the entire width of an 8-1/2 x 11 sheet of printer paper is filled when the 10-character/inch constant-width font is used. As a result, when file record length exceeds the permitted column width, records are folded to the next line. You should be aware that when you call **acctcms** single-line records are seldom possible.

The ASCII output produced when **acctcms** processes the *pacct* file is similar to the following listing. The entries shown are a fragment of the entire listing.

```
                                TOTAL COMMAND SUMMARY
COMMAND    NUMBER  TOTAL     TOTAL     TOTAL     MEAN     MEAN     HOG      CHAR      BLOCKS
NAME       CMDS    KOREMIN   CPU-MIN   REAL-MIN  SIZE-K   CPU-MIN  FACTOR   TRNSFD    READ
TOTALS     469     413.76     1.54     14.95     268.85   0.00     0.10     2471533   1566
express      1     172.46     0.18      0.57     933.89   0.18     0.32      115008     20
troff        2     157.25     0.32      0.82     494.49   0.16     0.39      418752     37
sh          95      15.27     0.18      3.83      82.52   0.00     0.05       44613    143
ls          14      11.10     0.09      0.45     121.13   0.01     0.21      182568     44
acctcms      7      10.30     0.08      0.21     123.62   0.01     0.40      142920     34
mount       56       8.72     0.12      0.29      73.71   0.00     0.40      194479     96
dmm         22     226.6     80.09      2.79      73.99   0.00     0.03       96242      7
pwget        1       3.47     0.04      0.09      90.43   0.04     0.41       32312     10
```

The first line in the output is a total of the entries in each column of the listing. Each of the fields of the records produced when the acctcms command is processed have the following significance:

`COMMAND NAME`

The name of the command the for which the statistics in each of the record fields have been accumulated. As can be seen from the `sh` entry in the fourth record from the top, 95 **sh** commands have been processed. Actually, this is the entry for all shell processes, regardless of their actual names. This is because only object modules are reported by the accounting system. The word `TOTALS` replaces the command name in the `COMMAND NAME` column in the first record written by **acctcms**.

`NUMBER CMDS`

This is the total number of time the command was processed during the accounting period. For example, *daycms* files are for the 24-hour period since the **runacct** script was last run.

`TOTAL KCOREMIN`

This value combines the amount of memory used and the length of time used. Memory is specified in Kilobyte blocks and time is specified in minutes. The computed values are obtained as follows:

kcore-min = [(cpu time in seconds) (mean size in kilobyte)]/ 60

`TOTAL CPU-MIN`

This is the total CPU time required to process the named command the number of times specified in the `NUMBER CMDS` column.

TOTAL REAL-MIN
: The total number of real-time minutes required to process the named command the number of times specified in the NUMBER CMDS column.

MEAN SIZE-K
: The average amount of memory in Kilobytes used to process the named command the number of times specified in the NUMBER CMDS column.

MEAN CPU-MIN
: The average amount of CPU time used each time the named command was processed the number of times specified in the NUMBER CMDS column. The computed value is obtained as follows:

mean cpu-min = total cpu-min/total number of commands

HOG FACTOR
: This is a measure of the relative amount of CPU time required to process the command the number of times specified in the NUMBER CMDS column with respect to the time required to process all commands. The computed value is obtained as follows:

hog factor = total cpu-min/total real-min

CHARS TRANSFD
: The total number of characters read or written when the named command was processed the number of times specified in the NUMBER CMDS column.

BLOCKS READ
: The number of file system blocks (one block is equivalent to 1 Kilobyte) read as the result of the process invoked by the specified command. The number of blocks read may not correspond with the number of characters transferred.

The acctmerg Command

The **acctmerg** command is called by the **runacct** script when **runacct** is executed by the **cron** daemon. The purpose of **acctmerg** is to combine the records in a number of daily files into a single total accounting file. For example, the total accounting records for a particular user login name and user ID are merged to provide a single group of records for that login name and user ID only.

The **acctmerg** command combines process, connect-time, fee, disk-usage, and queuing (printer) total accounting records in *tacct* binary or *tacct* ASCII format. The **acctmerg** command writes the results of record processing to standard output in the *acct* format, which is described in Table 15-4. The accounting file produced by the **acctmerg** command can have entries for as many as 18 columns (a complete *tacct* format). Column headings are printed only when you use the **-h** flag. Table 15-4 references the column headings by number, the column heading by name, and the purpose of the entry.

Table 15-4. Format for *tacct* Files (1 of 2)

Ref	Name	Purpose
1	UID	User ID. The integer value of the user ID.
2	LOGNAME	User login name. The user login name from the */etc/passwd* file.
3	PRI_CPU	Prime-time CPU usage. The number of seconds of prime-time CPU usage that was charged to the user.
4	NPRI_CPU	Nonprime-time CPU run time. The number of seconds of nonprime-time CPU usage that was charged to the user.
5	PRI_MEM	Prime-time memory K-core. Memory usage during prime time. This value expresses the amount of memory used and the elapsed amount of prime time during which it was used (K-core is the product of total CPU time in minutes and mean size of memory used).
6	NPRI_MEM	Nonprime-time memory K-core. Memory usage during nonprime time.
7	PRI_RD/WR	Prime-time read and write characters. The total number of characters transferred during prime time.
8	NPRI_RD/WR	Nonprime-time read and write characters. The total number of characters transferred during nonprime time.
9	PRI_BLKIO	Prime-time number of I/O blocks. The total number of I/O blocks transferred during prime-time read and write operations. The number of bytes in an I/O block is implementation dependent.
10	NPRI_BLKIO	Nonprime-time number of I/O blocks. The total number of I/O blocks transferred during nonprime-time read and write operations.
11	PRI_CONNECT	Prime-time connect duration. The total number of prime-time seconds that a connection existed.
12	NPRI_CONNECT	Nonprime-time connect duration. The total number of nonprime-time seconds that a connection existed.
13	DSK_BLOCKS	Disk blocks used. The total number of disk blocks used.
14	PRINT	Number of pages printed. The total number of pages queued to any printers in the system.
15	FEES	Special fee charge units. The number of integer units to charge for any special fee. This value is the one supplied when the **/usr/sbin/acct/chargefee** command is processed during the active accounting period.

Table 15-4. Format for *tacct* Files (2 of 2)

Ref	Name	Purpose
16	PROCESSES	Number of processes. The total number of processes spawned by the user during the active accounting period.
17	SESS	Number of logins. The number of times the user logged in during the active accounting period.
18	DSAMPS	Number of disk accounting samples. The total number of times during the active accounting period that the disk accounting command was used to get the total number of disk blocks listed in the DSK_BLOCKS column. Divide the value in the DSK_BLOCKS column by this number to obtain the average number of disk blocks.

Total accounting records are read from standard input and any additional files (up to nine) that you can specify with the *file* parameter. File records are merged according to identical keys, usually the user ID and user login name. To optimize processing performance, output is written in binary, unless the **-a** or **-v** flag is used.

Normally the **acctmerg** command is called from the **runacct** script, either to produce an intermediate file such as */var/adm/acct/nite/daytacct* when one or more source accounting files is full, or to merge intermediate files into a cumulative total file such as */var/adm/acct/sum/tacct*. The syntax of the **mergacct** command is:

/usr/sbin/acct/acctmerg -[ahipv] [*specification*] **-[tu]** [*file* ....]

The **acctmerg** command has 7 flags that tell it how you want it to process its input or produce its output. See the **acctmerg** command manpages for flag information.

The following example illustrates a field specification:

 # /usr/sbin/acct/acctmerg -i1-2,13,18 < dacct | tacct > output

The **acctmerg** command reads the columnar entries for UID(1), LOGNAME(2), DSK_BLOCKS(13), and DSAMPS(18) from the *dacct* file as input, merges this information as *tacct* binary records, and writes the resulting ASCII records to a file named *output*.

Inclusive field ranges can also be specified, with array sizes properly taken into account except for the *ta_name* number of characters. For example, **-h2-3,11,15-13,2** displays the LOGNAME (**2**), PRI_CPU (**3**), PRI_CONNECT time (**11**), FEES (**15**), PRINT (**14**), DISK_BLOCKS (**13**), and again LOGNAME (**2**), in that order, with the described column headings (**-h**). The default output produces all 18 columns (1-18 or 1-), which generates rather wide output records that contain all the available accounting data.

The acctprc1 Command

The purpose of the **acctprc1** command is to read process accounting records from standard input and login names that correspond to the user ID of each record. The records read from standard input must be in the *acct* format. Each record corresponding to the user ID is converted to ASCII and written to standard output. For each process, the record format includes the columns described in Table 15-4 on page 15-54.

To use the **acctprc1** command, standard input must be redirected from a process accounting file such as */var/adm/pacct*. Normally, the **acctprc1** command gets login names from the */etc/passwd* file. But when the system for which you are writing accounting procedures has users sharing the same user ID, you should reference a */var/adm/acct/nite* file instead of a *pacct* file. The syntax of the **acctprc1** command is:

 # /usr/sbin/acct/acctprc1 [*infile*]

Normally, the **acctprc1** command is called by the **runacct** script during its `PROCESS` state when **runacct** is executed by the **cron** daemon. When specified, *infile* contains a list of login sessions in a format defined by the *utmp* structure in the */usr/include/utmp.h* header file.

Login session records are sorted according to user ID and login name. When *infile* is not specified, **acctprc1** gets login names from password file */etc/passwd*. The information in *infile* is used to distinguish different login names that share the same user ID. The following command is an example of how **acctprc1** is used:

 # /usr/sbin/acct/acctprc1< /usr/adm/pacct > test.fle

When the **acctprc1** process is executed, the input is redirected from standard input to the *pacct* file. This file is processed and the resulting records are redirected from standard output to the *test.fle*. The output of **acctprc1** does not have column headings. The following is a typical **acctprc1** process output:

```
   0  root     0    1    17228   172     6
   4  adm      0    6    46782    46    16
   0  root     0   22   123941   132    28
9261  buzzbo   6    0    17223    22    20
   9  lp       2    0    20345    27    11
9261  buzzbo   0  554    16554    20   234
```

The acctprc2 Command

The **acctprc2** command takes records produced by the **acctprc1** command and summarizes them according to user ID and name, and then writes the sorted summaries to standard output as total accounting binary records in the *tacct* format. As with the **acctprc1** command, the **acctprc2** command is called by the **runacct** script during its `PROCESS` state when **runacct** is executed by the **cron** daemon. To produce a total binary accounting record of the ASCII output file *test.fle*, which is the file produced by the **acctprc1** command from the previous example, add the following line to an accounting script:

 # /usr/sbin/acct/acctprc2 /var/adm/acct/nite/daytacct

The resulting binary total accounting file, written in the *acct* format, contains records sorted by user ID. This sorted user ID file, is usually merged with other total accounting records when an **acctmerg** command is processed to produce a daily summary accounting record called */var/adm/acct/sum/daytacct*.

The prtacct Command

The **prtacct** command is actually a script called by the **prdaily** command that is used to format and print any total accounting file you specify. Records for the files produced when this command is processed are defined by a type *tacct* structure. The **prdaily** command is called by the **runacct** script during the `CLEANUP` state when **runacct** is processed by the **cron** daemon.

You can use **prtacct** to output any *tacct* formatted file to the standard output, usually the terminal. For example, you can output a daily report keyed to connect time, to process time, to disk usage, or to printer usage. To specify a title for the report, create a name for the heading parameter enclosed within single or double quotation marks. The syntax of the **prtacct** command is as follows.

 # /usr/sbin/acct/prtacct [**-f** *specification]* [**-v**] *file* ['*heading*']

Refer to the manpages for the **prtacct** command for flag information.

The *heading* argument specifies a heading for report members. The type *tacct* structure defines a total accounting record format, parts of which are used by various accounting commands. Members of the type *tacct* structure whose data types are specified as an array of two double elements have both prime-time and non-prime values. The type *tacct* structure has the same format as that described in Table 15-4 on page 15-54.

The **prtacct** script is used to produce the daily *daytacct* binary file in the */var/adm/acct/nite* subdirectory using the following command:

 # /usr/sbin/acct/prtacct daytacct 'DAILY DISK USAGE REPORT FOR *machinename'*

where *machinename* is the name of the system for which the *daytacct* file is produced. Because the **-f** flag is not used the default record format, which includes all the headings listed in Table 9-5, is used.

As an example, to produce a formatted daily usage report for a machine named MYMACHINE, enter the following command:

```
# /usr/sbin/acct/prtacct -f 1-2,13,18 daydacct \
'DAILY DISK USAGE REPORT FOR MYMACHINE'
```

In the previous example, **prtacct** is called to produce a daily disk usage file whose format includes the headings in Table 15-4 on page 15-54 with the specified numbers. The heading for the table is the character string between quotes. When a character string is not specified, no heading is written to the output file named *daydacct*.

The prdaily Script

The **prdaily** script is called by the **runacct** script and formats and produces the daily report ASCII file named *rprt.mmdd* in the */var/adm/acct/sum* subdirectory. One of these files is produced each day that the **runacct** script is called from the **cron** daemon. The invocation syntax for **prdaily** is:

 # /usr/sbin/acct/prdaily [**-l** | **-c**] [*mmdd*]

mmdd is the date of the report. When no date is specified, **prdaily** produces a report for the current day. When you specify *mmdd* the report is produced for the month specified by the *mm* month integer and day specified by the *dd* integer, but only for the current month because daily reports are accumulated only on a monthly basis.

The **prdaily** script combines information from several files into a single daily file. The reports and their source files are:

- A daily *reboots* report that receives its input from the daily reboots file in the */var/adm/acct/nite* subdirectory.
- A daily usage report that receives its input from the daily *lineuse* file in the */var/adm/acct/nite* subdirectory.
- A daily command summary report that receives its input from the daily *daycms* file in the */var/adm/acct/nite* subdirectory.
- A cumulative monthly total command summary report that receives its input from the daily *cms* file in the */var/adm/acct/nite* subdirectory.
- A last login report that receives its input from the *loginlog* file in the */var/adm/acct/sum* subdirectory.

Refer to the manpages for **prdaily** script flag information.

The lastlogin Command

The **lastlogin** command updates the */var/adm/acct/sum/loginlog* file to show the last date each user on the system logged in. Normally, the daily **runacct** script, which is called by the **cron** daemon, calls this command to write **lastlogin** information to the *lastlogin* daily report in the */var/adm/acct/sum/loginlog* subdirectory during its CMS state; however, the **lastlogin** command can also be entered from the command line. The **lastlogin** command uses the **printpw** command to get a list of all users whose name and user ID are stored in the password database file.

The printpw Command

The **printpw** command outputs the contents of the */etc/password* database file in ASCII format to the standard output. When **printpw** is called with no option, all usernames in the database are output.

The */etc/passwd* database file is accessed through the standard library function **getpwent**. On secure systems or on systems with an installed Yellow Pages service that have changed this library function, **printpw** produces the same information. The syntax of this command is:

 # /usr/sbin/acct/printpw [-acdgsu]

Refer to the manpages for flag information regarding the **printpw** command.

For example, to output the username, UID, and the login directory of all system users listed in the */etc/passwd* database file, enter:

```
/usr/sbin/acct/printpw -ud
operator:0:/usr/local/operator
smk:1394:/users/smk
fjc:1396:/users/fjc
cmb:1253:/users/cmb
rfm:1278:/users/rfm
jsr:1338:/users/jsr
seh:1340:/users/seh
```

Monthly Summary and Report Generating Commands

On the first day of the month the **cron** daemon calls the **monacct** script. This script moves three monthly files from the */var/adm/acct/sum* subdirectory to the */var/adm/acct/fiscal* subdirectory. When each of the monthly summary files has been moved to the *fiscal* subdirectory, the old daily ***rprt****mmdd* and ***tacct****mmdd* files are removed and new summary ***tacct****mmdd*, *cms*, and ***rprt****mmdd* are created. The files that get moved are:

***cms**mm* This file is total command summary file for the accounting period for the month preceding the referenced *mm* month integer. This is a binary file, which must be converted to ASCII using the **acctcms** command. To write the fiscal *cms* records to the standard output device, enter the following command:

/var/adm/acct/acctcms -s -a /var/adm/acct/fiscal/cms*mm*

where *mm* is 1 greater than the month whose records you want. This output can be redirected to another file, which provides you with a monthly command summary report.

***fscrpt**mm* This file is a total monthly ASCII report that is similar to the daily reports produced by the **prdaily** command. This report totals connect-session records from the summary *tacct*, the summary *cms*, and the summary *lastlogin* files for the month preceding the month referenced by the *mm* integer. You can print records to the standard output device with the **cat** command or redirect the output of the **cat** command to a file.

***tacct**mm* This file is a total accounting file for the accounting period of the month preceding the month referenced by the *mm* integer. This is a binary file, which must be converted to ASCII using the **prtacct** command. To write the fiscal ***tacct**mm* records to the standard output device, enter the following command:

/var/adm/acct/pracct /var/adm/acct/fiscal/tacct*mm*

mm is 1 greater than the month whose records you want. This output can be redirected to another file, which provides you with a monthly command summary report.

Turning Off System Accounting

There are two commands that turn off the system accounting functions: **turnacct off** and **accton**. These commands instruct the kernel to stop writing records to the active accounting files. Normally, system accounting is turned off by calling **shutacct** through **/sbin/rc0**. When the **shutacct** script is processed, a message stating that the accounting functions have been turned off is written to the */var/adm/wtmp* file. Then the **turnacct off** command is called, which tells the kernel that its active accounting functions should be shut down.

Permanent Shutdown

To permanently stop accounting on the system, perform the following steps:

1. Issue the **shutacct** command.

2. Remove the accounting commands that are called through the **/sbin/rc0** and **/sbin/rc2**. The easiest way to do this is to leave the script in **/sbin/init.d** as it is and to remove the links to it in **/sbin/rc0.d** and **/sbin/rc2.d**. If you want to enable accounting at some later time, you can add the links again.

3. Remove the following accounting function commands from the */usr/spool/cron/crontabs/adm* file:

```
5 * * * * /usr/sbin/acct/ckpacct

0 2 * * 1-6 /usr/sbin/acct/runacct

0 4 1 * * /usr/sbin/acct/monacct
```

4. Remove the following accounting function command from the */usr/spool/cron/crontabs/root* file:

```
0 3 * * 4 /usr/sbin/acct/dodisk 2>/var/adm/diskdiag
```

5. Remove the accounting files and subdirectories. Removal of the accounting files and subdirectories is straightforward. Removal is described in the next section.

Cleaning up Accounting Log Files

There are many accounting files in system directories and subdirectories. Refer to Table 15-2 for a complete list of these files and directories. You can remove most of the files listed in Table 15-2. However, you should retain the following files, directories, and subdirectories:

- /var/adm
- /var/adm/usracct
- /var/adm/savacct
- /var/adm/printer (and files)

Detecting and Replacing Bad Nodes 16

Occasionally, it may be necessary for you to replace a bad node board. For example, if one of the diagnostic applications identifies a questionable board, you should move it to a different location and determine if the error follows the board. If you are certain that the board is bad and you have a spare, you may want to install the spare and send the defective board to SSD Customer Service for repair.

This chapter discusses the detection, replacement, and return of bad node boards.

Detecting Bad Node Boards

Typically, you detect bad node boards in one of two ways:

- When running the Paragon System Diagnostics (PSD). Refer to the *Paragon™ System Diagnostic Reference Manual* for more information about these diagnostics.
- By checking the *BADNODES.TXT* file. This file is created by the **fscan** utility. The **fscan** utility is invoked by default at boot time and runs on the Diagnostic Station. **fscan** uses a system watchdog to monitor all nodes in the system and detect bad (or potentially bad) nodes. When a bad node board is detected, **fscan** lists the node in file *BADNODES.TXT* and then reboots the system. Refer to the *Paragon™ System Commands Reference Manual* for more information about **fscan.**

Replacing Bad Node Boards

SAFETY WARNING

Contact SSD Customer Service at one of the locations noted in the Preface before replacing a node board. You should only replace node boards under the direction of SSD Customer Service. Board replacement should *never* occur without our knowledge and consent.

To minimize the risk of electric shock and energy hazards, remove any watches, rings, bracelets, or necklaces that may be conductive before opening the rear door or any front panels. Also, to minimize accidental contact, immobilize one arm and hand by placing the hand in a pocket or behind your back before entering the service access areas behind the rear door and front panels. Assure that each movement is deliberate and do not contact any parts until a correct positive identification has been made. If there is any question in your mind about how to proceed in these locations, stop and refer to your SSD Customer Service representative before proceeding.

To replace a node board, perform the following steps:

1. Open the rear door of the cabinet that houses the bad node board. To open the door, insert a 3/16 inch Allen wrench in the Allen screw on the right-hand top of the rear door. Turn the Allen screw counterclockwise approximately 90 degrees to release the latch. You can then pull the door open.

2. Turn off the DC power to the mesh using the Off button located the power control panel. The power control panel is located inside the rear door of the cabinet, near the upper-rear of the cabinet (see Figure 16-1).

CAUTION

Guard against static when removing or exchanging nodes. Use a proper grounding strap.

3. To open the front door of a unit, move the lever (located at the bottom) counterclockwise 90 degrees.

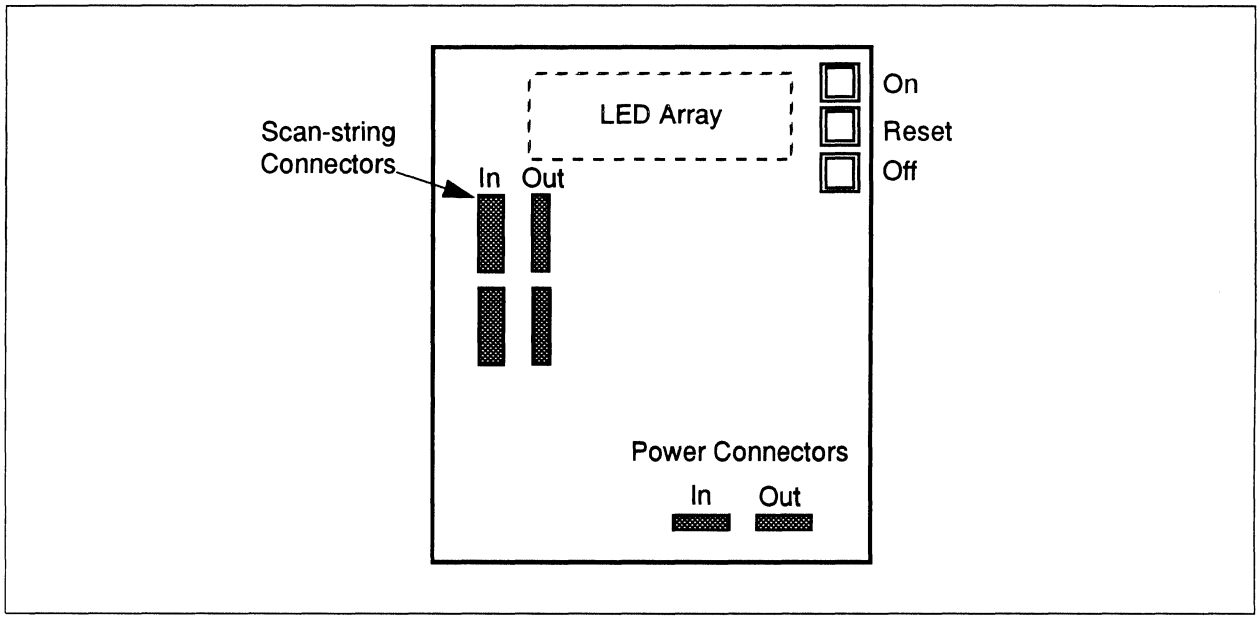


Figure 16-1. Turning off the Paragon™System

4. With a small screwdriver, loosen the screws at both ends of the malfunctioning board's front panel (see Figure 16-2 on page 16-3). The front panel screws are captive and cannot be removed from the panel.

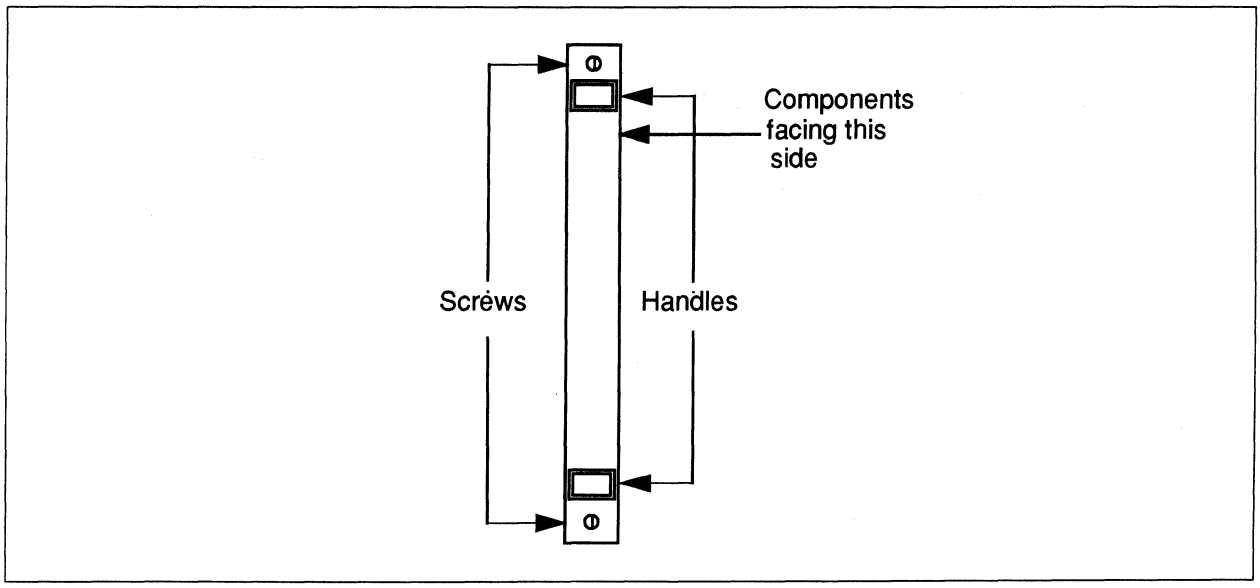


Figure 16-2. Node Board's Front Panel

5. Grasp both handles and push them apart. The board pops free of the backplane.

6. Set the defective board aside on an antistatic surface to be packaged and returned to SSD Customer Service for repair.

7. Insert a spare board into the tracks of the empty slot. Push the ejector handles to the center. Firmly seat the board with the palm of your hand.

8. With a small screwdriver tighten the screws in place. **DO NOT OVERTIGHTEN.**

9. If you cannot tighten the screws, the board is not properly seated. Remove the board, and visually inspect it for bent pins. If there are no bent pins, repeat Step 7, keeping the board straight while sliding it down the tracks of the empty slot.

CAUTION

If you discover any bent pins in the previous step, call SSD Customer Service for further instructions. Do not apply power to the mesh without first calling SSD Customer Service.

10. Apply power to the mesh using the On button on the power control panel (see Figure 16-1 on page 16-3).

11. Refer to the next section for information about returning a malfunctioning board to SSD Customer Service for repair.

Returning Bad Node Boards for Repair

When you contact SSD Customer Service (at one of the addresses noted in the Preface), have the following information available:

- The serial number of the computational unit. You can find the serial number inside the back door of cabinet 0 (the cabinet with the Diagnostic Station) on the swing-out power supply gate.
- Your shipping and billing addresses.
- Purchase order number (for billing purposes) if your warranty has expired.

Always contact SSD Customer Service *before* returning a board for replacement. You will be given a Return Materials Authorization (RMA) number, shipping instructions, and other important information.

Before shipping a board, make sure it is adequately protected as follows:

- The board should be placed in an antistatic bag within a padded shipping bag.
- Allow sufficient room in the shipping carton for protective padding such as flow pack, foam, and so on.
- Write, "Attn: Return Material Authorization (RMA) *number*" on the *outside* of the shipping carton (*number* is the RMA given to you by SSD Customer Service).
- Label the shipping carton FRAGILE.

Address and ship the carton to the address given to you by SSD Customer Service.

Marking a Slot as Empty

Occasionally, it is necessary to tell the system that a slot is empty when there is really a node board in it. If you do this, the system will not attempt to use the node in any way; no processes will be started on the node, and the node's scan line will be disconnected. You might do this if a node has failed and you cannot replace it at the moment.

To mark a slot as empty:

1. Log into the diagnostic station and the Paragon system as *root* (preferably in two separate windows).
2. On the diagnostic station, save a copy of the *SYSCONFIG.TXT* file in */usr/paragon/boot*. For example:

```
DS# cd /usr/paragon/boot
DS# cp SYSCONFIG.TXT SYSCONFIG.BAK
```

3. On the diagnostic station, edit the *SYSCONFIG.TXT* file in */usr/paragon/boot* to mark the specified slot as **EMPTY**. For example, to mark slot 7 as empty, locate the following line:

```
S  7 GPNODE AN00 32 MRC 04
```

   Change it to the following:

```
S  7 EMPTY
```

4. On the Paragon system, shut the system down:

```
# cd /
# shutdown now
# halt
```

5. On the diagnostic station, reboot the Paragon system:

```
DS# ./reset
```

When the system comes back up, the specified node will appear to be an empty slot (shown as a dash in the output of the **showpart** command), and any attempt to use the node will fail.

NOTE

The command **mergecfg** or **reset autocfg** will overwrite the change you have made in *SYSCONFIG.TXT*.

This occurs because **mergecfg** and **reset autocfg** create a new *SYSCONFIG.TXT* based on the system's current physical configuration. Because of this, if you mark a slot as **EMPTY** you should either physically remove the bad node board or replace it with a good node board as soon as possible.

When you have replaced the bad node, simply copy the saved *SYSCONFIG.TXT* file (*SYSCONFIG.BAK* in the example above) back to *SYSCONFIG.TXT* and reboot the system. (Exception: if the replacement node is not exactly the same as the node that failed, you should generate a new *SYSCONFIG.TXT* file with the command **reset autocfg** instead.)

Index

Symbols

C

D

E

F

K

L

M

Q

R

S

T

U

V

W

X

Y