ENHANCED ONYX SYSTEM V

ADMINISTRATOR GUIDE VOLUME I

Onyx Systems, Inc.
25 East Trimble Road
San Jose, California 95131
(408) 946-6330

Part Number [805-02667-001]

DISCLOSURES

ONYX Systems, Inc.
25 East Trimble Road
San Jose, California
95131
(408) 946-6330

Part Number: 805-02667-001

NOTICES

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

Onyx Systems, Inc. **MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.** Onyx Systems shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Onyx Systems assumes no responsibility for the use or reliability of software on equipment that is not furnished by Onyx Systems.

Enhanced ONYX System V

Administrator Guide Volume I

MAIN CONTENTS

TABLE OF CONTENTS

CHAPTER 1

CHAPTER 1

INTRODUCTION

Intent of this Guide

The Enhanced ONYX System V Administrator Guide is a reference manual. Its goal is to inform and instruct a UNIX system administrator on those functions that must be performed to establish, manage, and maintain a UNIX system successfully. This guide is but one of the resource "tools" available to help the system administrator in effectively performing this job.

How to use this Guide

This section describes how to use the Enhanced ONYX System V Administrator Guide.

The guide is divided into topics, each represented by a chapter with its own table of contents, and a list of illustrations if applicable. In addition, each chapter has a preface that describes the topic and the learning goal to be achieved.

All section headings appear in **bold type**. Sub-section headings, if any, are underlined.

For example:

This is a Section Heading

This is a Sub-Section Heading

The page numbering format reflects the chapter and page currently being addressed. (e.g., "1-1:" chapter 1, page 1).

Page "header" information contains the chapter and topic that are being discussed.

The terms "Caution" and "Warning" refer to important information that should be noted by the system administrator. **Caution** is used to inform the administrator of some special circumstance relative to the execution of a procedure or a UNIX command. **Warning** is used to warn the administrator that failure to follow a specific procedure or improperly using a specific UNIX command could result in damage to the system or loss of system data.

Where elaboration of a specific topic would extend beyond the scope of this guide, an explicit reference is made to the document which contains the needed additional information.

Procedural instructions are organized in numbered steps, each followed by the task to perform.

For example:

PROCEDURE: Title

1. Do this ...

2. Now do this ...

In those procedures where a command or action is requested to be performed, the command or action appears in **bold type**, immediately followed by the system's response.

For example:

PROCEDURE: Display a file's contents

1. Display the contents of a file.

   $ cat [file name] <cr>

   [The file's contents are displayed]

   $

Locating the Subject

To locate a subject, first look at the main contents page in the front of this guide, then select the chapter that best fits the question in mind. Next, advance to that chapter's table of contents and scan its list of subtopics for the one that specifically applies.

Description of Topics

The topics in this guide address what a UNIX system administrator needs to know and the functions that need to be performed.

Chapter 2 "The UNIX Operating System"

This chapter provides a general overview of the major features and structure of the UNIX system.

Among these topics are:

- File system's structure, files, directories
- Logical device structure
- Process control, structure, and memory management
- The user interface
- System accounting, protection and printer spooler
- Communication vehicles and methods
- Enhancements to UNIX
- UNIX system environments

Chapter 3 "Installing A New Version Of UNIX"

This chapter provides a list of general tasks and considerations for installing a new version of UNIX on a currently operating system.

Chapter 4 "Getting Started"

This chapter describes the tasks and considerations for initially setting up a UNIX system.

They include:

- Transferring data from tape to disk
- Loading the operating system into memory
- Verifying system integrity
- Moving between the various UNIX environments
- Configuring the system
- Backing up and rebuilding the operating system
- Required system accounts and directories
- Starting up and shutting down the system
- Documenting the system

Chapter 5 "The Daily Routine"

This chapter describes daily management and maintenance of a UNIX system.

The functions for this are as follows:

- System startup and shutdown
- Data storage resource management
- Saving and restoring user data
- Altering files and directories
- Creating new user accounts
- Managing user processes
- Communicating with the users
- Running system accounting
- System protection and security
- Handling user problems and resolving system errors

Chapter 6 "System Expansion"

This chapter describes how to add extra or different peripheral devices such as printers and disk drives to the system.

Chapter 7 "Handling System Problems"

This chapter describes how to define, isolate, and correct system problems.

These topics include:

- System diagnostic tools
- Crash procedures
- Emergency shutdown
- Who to call

Related Documentation

The following sources of information will be helpful to the system administrator:

- Enhanced ONYX System V USER REFERENCE MANUAL
- Enhanced ONYX System V PROGRAMMER REFERENCE MANUAL
- Enhanced ONYX System V ADMINISTRATOR REFERENCE MANUAL
- ONYX 6810 MICROCOMPUTER SYSTEM USER'S GUIDE for the system
- SOFTWARE RELEASE NOTICE

The first three manuals focus primarily on the "command" usage aspects of a UNIX system.

Periodically, changes are performed to the UNIX system software to enhance performance or correct some software anomaly. These changes, and any associated implementation instructions, appear in a "SOFTWARE RELEASE NOTICE" (SRN). These SRN's contain all the information necessary to effect the change. Each SRN thereafter should become a part of this guide.

Duties of the System Administrator

The duties of a System Administrator fall into two general categories:

- System activity management
- User management

System Activity Management

This category describes daily tasks which ensure continued system efficiency.

These tasks include:

- Starting up the system
- Monitoring system process activity
- Managing the data storage
- Altering system attributes
- Managing system accounting
- Handling system problems

In addition to these functions, the system administrator should also maintain historical records pertaining to system configuration, system problems, data backups, system usage, and system security.

User Management

This category can be divided into three main elements:

- Educating users
- Informing users
- Handling user problems

Many problems can be avoided if the user has a basic understanding of how to use the computer. The responsibilty for providing this knowledge generally rests upon the system administrator.

Some of the areas in which the user should have knowledge include:

- How to "log" into the system
- How to create, edit, and print files
- How to "move about" through the file system structure
- How to maintain system security
- When and who to ask for help if a problem arises

Users should be kept apprised of changes or events involving the system which might have an impact on their work. Some of these events include: reconfiguration of the system, addition of special hardware or software features, scheduled maintenance, and system problems.

The UNIX system, as will be seen in later chapters, provides a number of ways to accomplish this task easily.

Users can and will, in the course of learning to manipulate the UNIX system, "pilot themselves into a black hole, from which there is no obvious escape." "Pilot errors," as they are often called, may involve: improperly exiting from an edit session, attempting to access a restricted file, or using incorrect command syntax.

These types of problems are generally easy to solve. However, it is important for the system administrator to listen to the users' problems because their interaction with the system is more constant. Users can inform a system administrator of subtle changes in system performance or actions that might indicate a more serious underlying problem.

Setting Up Procedures

This section provides certain considerations for setting up a UNIX system.

First Encounter

Upon receiving the system, the system administrator should verify that all the items necessary to effect the installation are present. Onyx provides a "checklist" just for this purpose. All discrepancies should be noted, and the proper people should be notified in order to obtain or replace missing or damaged parts.

Familiarization

Before attempting to use the UNIX operating system, the system administrator should become acquainted with the various manuals and documentation provided. Skipping this step can create unnecessary problems and delays in the installation of the system.

If there are any procedures that are not understood, the system administrator should contact the technical support staff of the sale organization from whom the system was purchased for help.

Initial System Loading

Upon configuring and activating the system hardware, the system administrator should perform any tests and diagnostics that may be suggested by the instructions.

Once satisfied that the hardware is functional, the system administrator should start transferring the UNIX operating system from tape to disk.

Once the transfer is completed, further tests can be performed to verify that no errors occurred during this process and that all the necessary files and directories are intact.

System Configuration Planning

Planning the system configuration involves determining the following:

- The number of user accounts to be installed
- What peripheral communication devices are to be supported
- What other special peripheral devices are to be supported
- Whether or not system accounting will be used
- What system protection and security measures will be used
- Disk file system initialization
- Any other special changes/additions needed to accommodate specific system requirements

TABLE OF CONTENTS

CHAPTER 2

LIST OF ILLUSTRATIONS

CHAPTER 2

THE UNIX OPERATING SYSTEM

Preface

This chapter describes the structure and major feature content of the UNIX operating system. The goal of this chapter is to help the system administrator achieve a fundamental understanding of this structure, its features and tools, as they apply to the job of UNIX system administration.

It should be noted that if you're already familiar with the UNIX system, then you may want to proceed directly to Chapter 4.

UNIX Overview

The UNIX operating system was born out of the need to have a computing environment in which programmers could comfortably and effectively pursue their programming research endeavors. This was the goal the Computing Science Research Group at Bell Laboratories set out to achieve, and did.

UNIX has undergone many changes since that original version in 1968. Over the past seventeen years, it has been enriched in flexibility and power; and it has grown in popularity, expanding its followers far beyond the academic sphere of the universities and firmly into worlds of business and government.

Now that you have an idea of **How** the UNIX operating system came into being; let's discuss **What** the UNIX operating system is.

The UNIX operating system is:

- An INTERACTIVE SYSTEM
- A MULTI-TASKING SYSTEM
- A MULTI-USER SYSTEM

Interactive System

By this it is meant that a user enters commands, and the system obeys these commands and displays appropriate responses.

Multi-tasking System

This means that a user can instruct the system to perform a number of tasks (called "processes") at the same time, freeing the user to concentrate on a new set of tasks.

Multi-user System

This attribute allows more than one user to use the system at the same time. It comes as a natural consequence of the "multi-tasking" system just described: the system can attend just as easily to multiple users at the same time as it can to multiple processes at the same time.

The Major Parts of UNIX

The UNIX operating system, though complex in appearance, can be divided into three major parts: the Kernel, the File System, and the Shell.

The **Kernel** manages all device resources of the system such as the disk drive, tape drive, terminals, communication lines and any other devices.

The **File System** provides organization for the data that will be created by users and stored on disk or tape.

The **Shell** makes UNIX an interactive system. The shell listens and interprets the commands a user enters from a terminal.

In UNIX, a command is often called a "utility" because it performs a useful set of tasks for a user. Concurrently, the term "utilities" refers to all the UNIX commands collectively. These terms appear often throughout this guide.

Major Features

The following text provides a conceptual look at the major attributes of the UNIX system.

File Systems

A file system represents an allocated area of disk storage for which its size and boundaries have been established. Under normal operation, a system's disk storage may be partitioned into many file systems. This is desirable because UNIX treats each file system and the contents therein as a separate entity, capable of being "removed" from the system (transferred between tape and disk) without disturbing any other file system or its contents.

However, the desirability of this feature does not come without some restrictions:

- File system contents cannot be truncated into another file system.
- File systems cannot bridge between physical disk drives.

File systems within the UNIX system are accessed through an assigned directory "name" (e.g., F1, F2 etc.).

Figure 2-1, "Logical File System Structure," illustrates the various components involved, and shows their relative order of occurrence within a file system.

Let's briefly explain the function each of these components serves.

The **Boot Block** is the very first block (block 0) of a file system. It is where the information needed to "bootstrap" a UNIX system would be put. In general, this bootstrap information exists only in one file system of a UNIX system. Therefore, the boot block for the other file systems has no real significance.

The **Super-Block** (block 1) of every file system contains the major pieces of information about that file system such as its size in blocks, the file system name, number of blocks reserved for i-nodes, the free i-node list and the start of the chain of free blocks (those blocks that are available for use).

"What is an **i-node**?" An **i-node** is an object which contains information that describes a file on the UNIX system. Therefore, every file on the system will have an associated i-node. In the UNIX system, i-nodes are represented by a number.

The number of i-node blocks depends upon the total number of blocks established by a file system.

Along with the descriptive information, each i-node contains "pointers" to the location of the blocks that comprise that file. These pointers are very important because in writing data to the disk, the disk will attempt to place the data contiguously (next to each other); however, if it cannot do so, it will place the data wherever it can find a free space within that file system. Without these pointers a file could never be reassembled.

Note: each "logical" block reported by the UNIX system utilities is 512 bytes in size; however, "physical" blocks (disk) are 1024 bytes in size.

When UNIX is installed on the Onyx computer system, four file systems are initially configured.

The four file systems are:

- c0d0s0 -- the reserved area
- c0d0s1 -- the "root"
- c0d0s2 -- the "swap" device
- c0d0s3 -- the "usr"

The **c0d0s0** or reserved area file system is 1188 blocks in size and contains the system bootup data and disk configuration information.

The **c0d0s1** or root file system is 8,000 blocks in size and is where UNIX resides.

The **c0d0s2** or swap device file system is 3,000 blocks in size and serves as the temporary storage space for process data that is "swapped-out" during system operation.

The **c0d0s3** or usr file system contains all additional user commands and their libraries.

Files and Directories

Within the UNIX scheme of data organization, two more instruments are employed to control and define this organization further: files and directories.

A **file** can be thought of as a "box" in which data is placed, and then labeled with a "filename." The most important part of this box is its filename because it is through the reference of this filename that you can address and manipulate the data contained therein.

A **directory** is simply a file which contains information about other files or other directories called "sub-directories."

The overall relational structure of files, directories, and file systems has been described as an "upside down tree" having its base or "root" at the top and its branches reaching out and downward from the root.

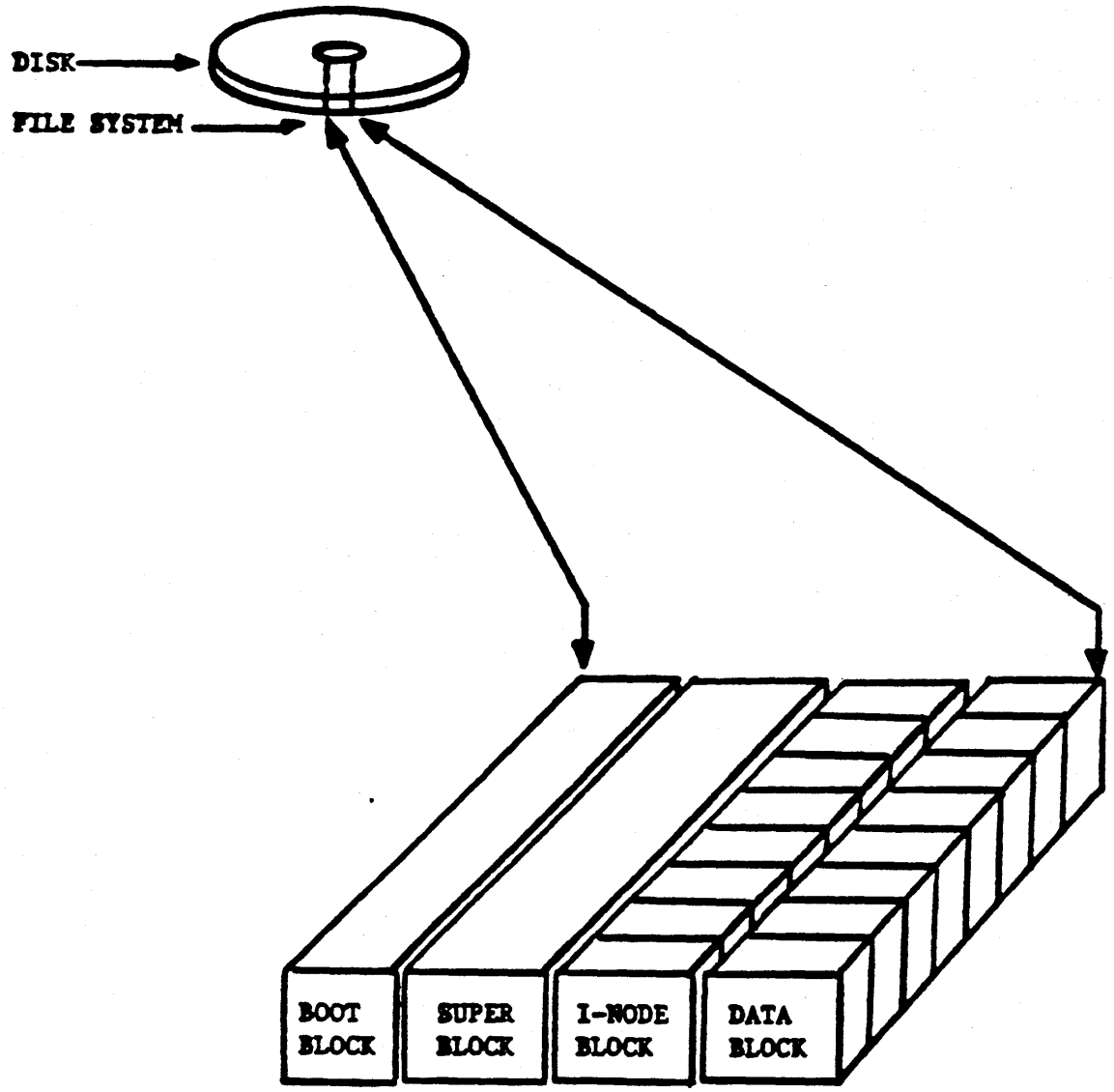


Figure 2-1 Logical File System Structure

The method used by UNIX to traverse these branches and limbs is called a **pathname**. Pathnames, like road maps, provide directions for how to get from one place to another. For example: A user wants to print a file named "apple," and apple is contained in a directory named "fruit," for which both apple and fruit reside in a second file system named "F2."

The pathname in this case looks like:

[command] /F2/fruit/apple

The first **slash** ("/") in the pathname is the symbolic representation (name) for the "root" file system, which is always the starting point. Subsequent slashes instruct the system to continue searching for the file from the last named directory in the path.

As you examine the file system, certain files may appear which possess suffixes of ".o," ".h," and ".c." These suffixes indicate the file is of the binary object type, a "C" language Header-include type or a "C" language source file, respectively.

Note: other suffixes such as: ".a," ".f," ".cbl," and ".s" indicating file types of archive, FORTRAN, COBOL, and assembler respectively may also appear.

Logical Device Structures

A "logical device" deals primarily with the kernel-to-user communications portion of the UNIX system. Three major elements are involved in this communications process: the logical device file, the device driver, and the device electronics.

A **logical device** is simply a special file which contains information that the kernel will use when a user is communicating with a peripheral device such as a disk drive, tape drive, terminal, or some other device.

A **device driver** is a program deeply embedded in the kernel which provides the intelligence necessary to communicate directly with the "device electronics."

Figure 2-2 , "Logical Device Interaction Scheme," examines the relationships among each of the components involved, and shows the flow of interaction between a user and a device.

Logical device files can be of two types: "blocked I/O" and "character I/O."

The "type" of a logical device will depend upon the requirements of the device driver it is interfacing with. Both **blocked** and **character** refer to the manner in which the device driver accepts data, and I/O is short for Input/Output.

In **blocked** mode, data will be assembled in groups of a specific size prior to being transferred to the intended device. In UNIX, each block typically includes 1024 bytes of data. One of the peripheral devices which uses blocked I/O is the disk drive.

In **character** mode, data will be dealt with on a character-by-character basis. A typical peripheral device which uses this type of I/O is the terminal.

One of the reasons that some devices work in blocked mode, while others use character mode, relates to the action that is going to be performed on or to the data. If the action is one of moving large amounts of data from one place to another, where no interpretation of this data is required, the blocked mode is employed for speed and efficiency. Where the interpretation of data is necessary, character mode is used.

Note: some devices utilize both of these modes of data transfer. In addition, the device files concurrent to these modes are contained in sub-directories in /dev (e.g., "/dsk," for blocked disk mode and "/rdsk," for character disk mode.)

Logical device files in the UNIX system are contained in the /dev directory. Generally, they may appear like this:

```
crw--w--w- 1 root     0     8, 1 (date/time) filename

Where:
          c = mode: "b" if blocked "c" if character
 rw--w--w- = file permissions field
          1 = number of links to other files
       root = the owner
          0 = group it belongs to
         8, = major device number
          1 = minor device number
```

The **major** and **minor** device numbers determine what class of device driver (major) is to be used and the specific device (minor) itself. The other fields mentioned are discussed in detail in subsequent chapters.

The mnemonic names used for device driver files in this version of UNIX are quite different from those used in prior released versions. For example, in UNIX System III the root file system device file is named "rp1;" however, in this version it is named "c0d0s1."

The naming convention format used in this version is as follows:

```
c0d0s1
```

Where:

- c = the hardware controller device being requested. In this case, "c0" is the disk controller.
- d = the device driver being addressed. In this case, "0" indicates the first disk drive.
- s = the file system being addressed. In this case, "1" is the root file system.

In addition, there are other differences pertaining to device driver files and how to address them which are discussed in subsequent chapters of this guide.

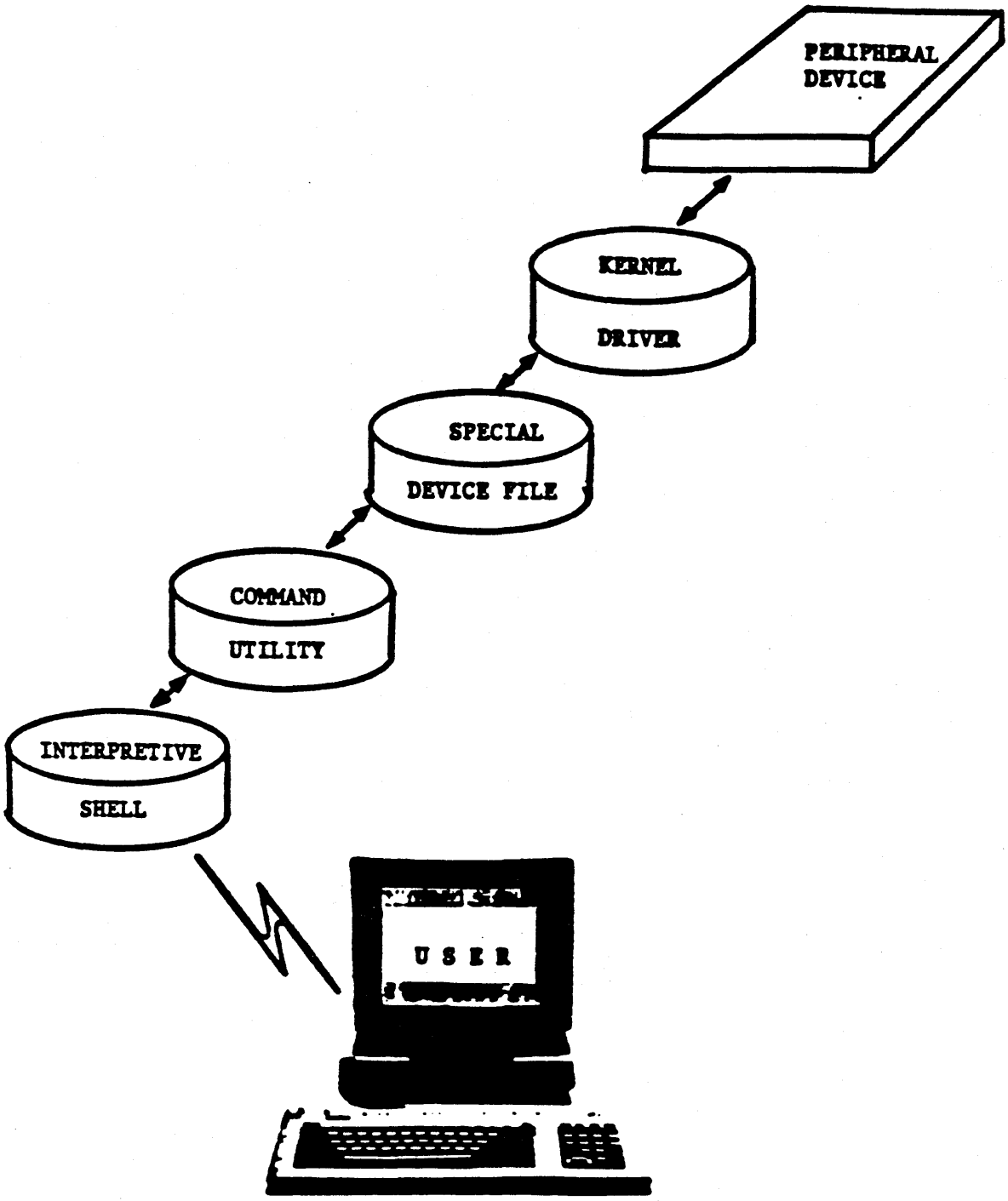


Figure 2-2 Logical Device Interaction Scheme

Process Structure

As described earlier, the UNIX system is a multi-tasking operating system. In UNIX, processes can be executed in one of two environments:

- Foreground
- Background

A **foreground** process is one for which the interaction between the user and the invoked process is maintained until the task has concluded and the results have been conveyed. Therefore, a user can execute only one process in the foreground at a time.

A **background** process is one that, once invoked and placed into this environment by a user, will execute independently and require no further interaction between the user and the process. Therefore, a user is able to perform multiple tasks at the same time. A process is placed in the background environment by appending an **ampersand** (&) to the end of a command line.

The system keeps track of these processes and who owns them by assigning to each process a **process identification** (PID) number. This PID number is conveyed to the user who originally invoked the process; it is prudent for the user to make a note of it in the event it becomes necessary to terminate the process manually.

Another aspect of a background process is that it may draw upon the resources of another process if required to complete its task.

The relationship between a background process which calls another process is a **parent** (the calling process) and **child** (the called process) relationship.

A parent process may have many "children" tied to it. However, a parent process cannot conclude until all of its children processes have completed their tasks.

Process activity is recorded and constantly updated by the UNIX system. It can be viewed with the **ps(1)** command. The use of this command and the management of processes are discussed in subsequent chapters of this guide.

Memory Management

In general, the computer memory serves as the system's workplace. Data is brought into it, manipulated, and then returned to where it came from or to some other designated place. How well a computer system performs is, in part, dependent upon how fast its electronic memory operates and how efficiently the memory is managed.

Memory management has both software and hardware connotations. In the hardware sense, "management" applies to the control of the electronic mechanism that transfers data to and from memory and keeps track of where data is placed.

The software aspect pertains to making the decisions involving **what** data is to be transferred and **when** this transfer is to take place.

This discussion is confined to the general software aspects of memory management.

In the UNIX system, this management task takes the form of a priority scheduling and a memory allocation control scheme. When a process is invoked, the UNIX system will scan a table containing a list of all the processes currently in progress and those waiting to be performed. The system will then add this new process to that list. Processes are generally serviced on a "first-in-first-out" (FIFO) basis. However, there are some exceptions which pertain to special processes that help maintain system integrity in terms of: updating the system's knowledge of new or altered files and directories, and attending to processes which "listen" to a user's terminal. These types of processes have high priority, and therefore can supersede this FIFO scheme.

The priority assigned to a process, as described above, is primarily controlled by the system; however, a user can manually assign a process priority number by using the **nice(1)** command.

The second part of memory management mentioned above is the "memory allocation" control scheme.

Before a process can be performed, the UNIX system must determine if there is enough memory available to accommodate all the data needed by the process to accomplish its job. The memory management programming in the kernel makes this determination. The actual task of memory resource allocation is performed by the "swapper" programs.

The **swapper(s)**, "swapin" and "xswap," handle those situations where a reallocation of the memory resource is needed to accommodate a process whose priority deems it is to be serviced immediately. The **xswap** program temporarily moves another process's data from memory to disk storage, and then places the priority process's data into that memory location. The **swapin** program puts back the swapped process's data into memory as soon as the priority process has completed.

User Interface

What is meant by a "user interface?" Simply this: it is the vehicle through which a user's requests are communicated and interpreted by the computer operating system.

In a broad sense, the computer terminal could be considered a user interface because it is one of the vehicles used to communicate with a computer system. However, the discussion is limited to the UNIX software aspects that interplay and support interaction between user and computer.

As previously mentioned, the **shell** is thought of as being one of the major components of the UNIX system. (See Figure 2-3, "Where the Shell fits in the UNIX System.")

The **shell** is a program which acts as an "interpreter," listening to those requests that a user enters from a terminal, and translating them into actions on the part of the kernel or some other system program. However, without limiting the generality of the foregoing, the shell possesses much power. It has many special attributes that provide flexibility for a user when working within the UNIX system.

For instance, the use of pathnames and the placing of a process in "background" are functions of the shell. The ability to direct a process's input or output is an attribute of the shell; and the construction of simple or complex programs without needing to be a "programmer" can be accomplished through the shell.

Examples of shell programming, also known as "shell scripts," can be found throughout the UNIX system. Programs such as "rc," ".profile," and "shutdown" are shell scripts created to perform various system tasks automatically.

The point here is that the system administrator should take the time to become familiar with this particular UNIX facility.

There are two shell environments available under this UNIX system:

- The native Bourne Shell (sh)
- The Berkeley C-shell (csh)

The Bourne Shell was so named by its creator, Mr. S. Bourne of Bell Laboratories; the C-shell was so titled by the group who developed it at the University of California at Berkeley. The differences between these shells lie in feature content.

The system administrator can find additional information about the shell **sh(1)** and the C-shell **csh(1)** in the Enhanced ONYX System V USER REFERENCE MANUAL.

Another aspect under the topic of user interface is the user "account." An **account** is basically a software representation of a person who has been granted access to a UNIX system.

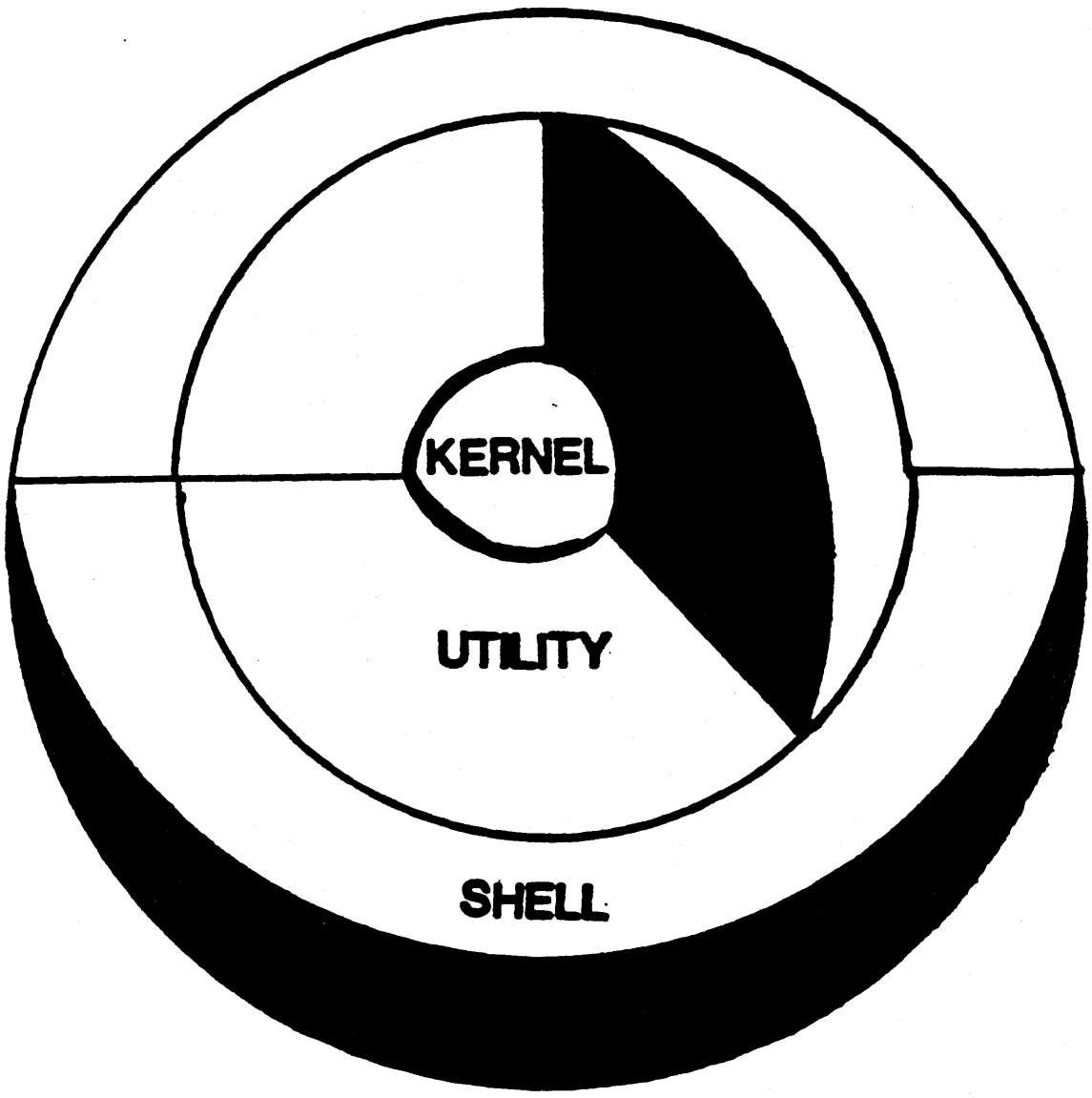


Figure 2-3 Where The Shell Fits In The UNIX System

Like people, accounts have unique personalities which describe and distingish them from other accounts.

In the UNIX system, an account can possess the following attributes:

- Login name
- Encrypted password
- User identification number
- Group identification number
- Accounting name
- Login directory
- Program name

Printer Resource Scheduling

Printer resource scheduling feature allows users to "queue" or schedule files to be printed by a printer, freeing the user to go on to some other work.

In the UNIX system, the printer scheduling function is performed by a program called **lp(1)**. Lp can be programmed to perform a number of special functions:

- Inform the user upon completion of the job.
- Remove the printed file after completion.
- Make a copy of the printed file after completion.
- Cancel any queued files waiting to be printed.

In addition, there are seven programs that allow the administrator to manage, control, and configure certain aspects of the printer scheduling facility. These seven programs are:

- accept(1M) -- allows printer requests to be accepted.
- reject(1M) -- inhibits printer requests.
- lpstat(1) -- displays status information concerning printer scheduler.
- lpsched(1M) -- activates the printer scheduler.
- lpshut(1M) -- deactivates the printer scheduler.
- lpmove(1M) -- moves printer requests to another printer device.
- lpadmin(1M) -- configures the printer spooling system.

The manipulation and execution of these programs are discussed in subsequent chapters of this guide.

System Accounting

System accounting keeps track of **who** is using the computer resource, **how much** of the resource is being used, and **how efficiently** the resource is working. It is the method used by companies who offer the use of their computer resource to others (time sharing), to determine how much to bill the other "party" for this service.

The system accounting facility in the UNIX system gathers statistical data on and monitors the activities of:

- Disk access occurrence
- Disk storage utilization
- Process execution time
- Process usage
- Frequency of user/computer access

However, system accounting, when allowed to run unnecessarily, can accumulate volumes of data which will chew away at the available disk storage and degrade the performance of the system. Therefore, the system administrator should take this negative aspect into account when deciding whether or not system accounting is necessary in the application.

System Protection

System protection conjures up many interpretations. According to some people, it has been said to encompass every area from physical security of a computer facility to the protection of the data itself. And in fact, these and all the areas which fall between them are aspects of system protection! However, this discussion of system protection in the UNIX system is limited to the following:

- User access control
- User data archive

User Access Control

This subject can be divided into two basic levels: system entry, and file access.

System entry involves the control over who shall be granted access to the system resource. The UNIX system achieves control over this aspect with accounts and passwords.

As mentioned above, accounts are used to identify a valid user by name in the login process. However, since many people possess the same name, a second method of identity is used -- the "password."

A password provides that unique characteristic which distingishes one user from another or determines whether or not they are valid users at all.

Passwords, in the UNIX system, are set by the users themselves. To be effective, a password should have the following attributes:

- It should be UNIQUE.
- It should be able to be REMEMBERED.
- It should NOT BE SHARED.

In addition, passwords are "encrypted," ensuring that they do not appear in their natural form anywhere in the system that is accessible to a user.

The next level under user access control is "file access."

UNIX offers three basic methods a user can use to control access to files: file "permissions," "restricted" shell, and file "encryption."

A file's **permissions**, also known as its "mode," describe what type of access is allowed, and by whom. Each file created will have a corresponding field which exhibits its permissions.

For example:

```
            [prefix]rwx-rwx-rwx  [other information] [filename]
                      :   :   :
                    [1] [2] [3]
```

Where:

```
       1 = OWNERS permissions field.
       2 = GROUP permissions field.
       3 = WORLD or other users permissions field.
       r = means READ permission with an octal value of "4."
       w = means WRITE permission with an octal value of "2."
       x = means EXECUTE permission with an octal value of "1."
[prefix] = may either be a "b" for blocked, "c" for character
           or "d" for directory.
```

The next tool used to restrict and control access is the **restricted** shell -- **rsh(1)**. This feature can limit a user to a given defined group of UNIX commands. It should not be thought of as a separate shell, but merely another tool the shell **sh(1)** provides. This capability is especially useful in controlling access for "casual users:" those who require only occasional access to the system.

The last tool mentioned is file **encryption.** The UNIX system maintains a facility for which a user can provide ultimate protection of a file's contents.

This encryption facility is based upon the National Bureau of Standards "Data Encryption Standard" (DES). It is implemented with the **crypt(1)** command, and its use and options are described in detail in the Enhanced ONYX System V USER REFERENCE MANUAL.

Note: the **crypt(1)** utility is not available to international users.

User Data Archive

Protecting the user's work is an important activity! It requires maintaining a periodic copy of the user's work or other system information that is valuable and necessary to operation of the facility.

When the system administrator should make system copies (backups) depends upon how much data is accumulated over a given period of time and its value.

The methods available under UNIX are as follows:

- tar(1) -- a tape archive utility
- finc(1M) -- a fast, incremental file system backup utility
- frec(1M) -- a fast, incremental file restore utility
- volcopy(1M) -- a file system copy utility, with label checking
- dd(1) -- a device-to-device data transfer utility
- cpio(1) -- a data copy input-output program

Chapter 5, "The Daily Routine," describes when backups should be performed, and what method is most applicable.

Communications

The UNIX system offers and supports a wide variety of methods and vehicles for communicating information, locally or remotely, to users or other computer systems.

The four most commonly used methods of communication are:

- cu(1C) -- call UNIX
- uucp(1C) -- UNIX to UNIX copy
- mail(1) -- an electronic mail facility
- news(1) -- an electronic bulletin board

These methods can be grouped into two general categories: those which are primarily used for conversation with or data transfer to another computer system; and those which are used chiefly to keep the user informed.

Conversation and Data Transfer -- cu uucp

The cu(1C) program utility provides the basic requirements needed to establish communications through a peripheral communications device such as a modem.

Cu can be used to log in and transfer data to another computer system operating under UNIX, or sometimes, to a different operating system.

The uucp(1C) program is designed to accommodate the transfer of large volumes of data between computer systems in a controlled environment. Like cu, uucp can interface and support communications through the above mentioned peripheral devices. However, uucp is far more sophisticated in that it carefully monitors all associated activity and provides an "audit trail" in the event that problems arise.

Keeping the user Informed -- mail news

The mail(1) program allows the sending and receiving of letters from individuals or groups of users.

The news(1) program, on the other hand, is primarily used to inform users of a coming event such as: "the system is going to be down for maintenance." News does not involve any interactive dialogue between users.

All of these utilities, plus others that involve communication between users, are discussed in subsequent chapters of this guide.

Onyx Changes and Enhancements to UNIX

As stated earlier, the UNIX system has undergone many changes since its inception. This section describes some of these enhancements and their values.

Onyx Changes

To improve system efficiency and performance, Onyx has enhanced, and/or developed and implemented the following features in this UNIX system:

- Record locking
- Scatter memory allocation
- Shared text
- Auto configuration

Record Locking

As already explained, one of the features of the UNIX system is its ability to allow multiple users to access and alter the contents of a file at the same time. However, this ability to "share" a file's contents presents a potential problem. How can you be assured that alterations to a file's contents are safely made and saved, while that file is being shared?

Users can be assured of this by "record locking," a means which protects that portion of a file's contents undergoing changes, and prevents any intervention until the changes are effected.

Record locking permits multiple user processes to "lock" or secure segments of a file's contents while that segment is being altered.

The Onyx implementation of record locking also corrects situations where multiple user processes attempt to access the same segment at the same time, or a segment that is already locked. Record locking is transparent to the user.

Scatter Memory Allocation

As discussed previously, memory management has a significant effect upon the system's overall efficiency and performance. Typically, when a block of data is loaded into memory, it occupies a contiguous block of memory space. This scheme of memory management is adequate. However, when system usage becomes heavy and contiguous memory space is unavailable, data to be written into memory must be delayed. These delays decrease the performance of the system. To counteract this condition, Onyx has implemented **scatter memory allocation.** Scatter memory allocation is the ability to split a process into fixed length pages and load them into any available memory space without regard to contiguity. This memory managment feature maintains system efficiency and performance during those times when the system is in heavy use.

Shared Text

Concurrent to **scatter memory allocation**, Onyx has enhanced its memory management scheme further with **shared text.** This feature allows a single copy of a program to reside in memory and be "shared" (accessed) by more than one user at the same time. This sharing process significantly reduces the occurrence of swapping out users' processes when system activity is heavy. In addition, when swapping does occur only the stack/data space needs to be swapped; generally, there will be enough physical memory space available to accommodate the text/stack/data without having to swap out any other user.

Auto Configuration

Auto Configuration is the means by which the UNIX operating system adjusts to the system it is running on automatically! This feature, an Onyx created utility, tailors the UNIX kernel and associated support files using the information passed to it from the system firmware (PROM) thereby making start up much easier!

Enhancements to UNIX -- The Berkeley Utilities

By now, you should be familiar with the references of "Berkeley" and the "C-shell." The C-shell is but one of a variety of useful utility programs developed by the University of California that have been added to the standard UNIX system.

Some of the other utilities in this group, collectively known as the "Berkeley Enhancements," are:

- tset --- a program that configures terminals based upon "termcap"
- ex --- line oriented text editors
- termcap --- a database describing the personalities of terminals
- mail --- an electronic mail system
- more --- a program that displays the contents of a file a page at a time

Much has been written on these utilities, and their descriptions can be found in the Enhanced ONYX System V USER REFERENCE MANUAL.

UNIX System Environments

The UNIX system offers a choice of four system work environments:

- Standalone
- Single-user
- Multi-user
- Super-user

The following describes the general purpose of each environment, what general functions can be performed, and what functions should not be attempted under these environments.

Standalone Environment

The **standalone** environment, that which is evoked upon powering-up and "booting" the computer system, is actually outside of the UNIX system. It is established primarily through programming which resides in Read-Only Memory (ROM) devices in the computer system.

The system indicates that it is in the standalone (shell) environment by displaying the **SHELL$$** prompt message on the console terminal screen.

Note: only the console terminal is active in the **standalone** mode.

The primary purpose of **standalone** is to provide a working level which supports the diagnostic testing of the system's hardware elements, in addition to allowing the system administrator to effect repairs to the UNIX system which cannot be performed under the other environments.

These **standalone shell** programs are as follows:

- cp -- makes a copy of a file
- cat -- displays the contents of a file
- diskconf -- configures the disk file system
- ed -- edits a file
- format -- formats and prepares a disk drive
- fsck -- checks a file system for errors
- fsdb -- debugs a file system
- ln -- logically links files together
- ls -- lists the contents of a directory
- mv -- moves, copies and or renames files
- mkdir -- makes a directory
- mkfs -- makes a file system
- mknod -- makes a directory entry and corresponding i-node for a special device file
- od -- displays the octal representation of a file's contents
- rm -- removes files
- rmdir -- removes a directory
- spare -- spares defective disk sectors
- sparelist -- displays the list of spared disk sectors
- ustat -- displays special data about a file system

```
*************** WARNING ***************
*                                       *
* The mkfs, format, and spare programs  *
* destroy data on the disk!             *
*                                       *
*****************************************
```

Single-user Environment

Single-user is the next operational level after the UNIX system bootup has been completed. The system indicates that it is in single-user by displaying a **pound** symbol (#) prompt character on the system console screen.

Single-user, as its name implies, means that only one user, the console, can interact with the system.

This environment, when invoked, will allow the system administrator to perform functions relating to the following:

- System preparation
- System integrity verification
- Data backup and restoration

System preparation includes: setting the system's "clock" to the appropriate date and time; removing, copying and moving files; altering system configuration parameters; modifying, adding or removing system user accounts; and so on.

System integrity involves determining whether or not a file system's organization has been corrupted due to an event such as a sudden loss of electrical power to the computer system.

Data backup and **restoration** is another function performed in this environment. It is prudent to take backups in this mode because users do not have the ability to make any further changes; therefore, the data is considered stable.

While a system administrator is working in single-user mode, there are a few precautions to be considered.

First, many of the programs that are designed to protect the system from loss of data are not running. Therefore, it is wise to perform only necessary tasks, then proceed to the normal operational environment -- **multi-user.**

Second, the console terminal is the ONLY terminal that can interact with the system; therefore, do not attempt to send information to another user.

Multi-user Environment

The **multi-user** level is considered to be the normal operational environment. Multi-user mode is initiated by the user entering an **init 2** command on his/her terminal which invokes the **init** program residing in the /etc directory.

The system indicates to all users that it is in multi-user mode by displaying either a **dollar** sign ($) or a **percent** sign (%) prompt character on all users' terminals.

Upon invoking the multi-user environment, certain programs are automatically set into operation:

- bcheckrc
- brc
- rc
- cron
- openup
- lpd

The following is a brief description of the purpose of each program.

The **bcheckrc** program allows the user to perform a file system check (**fsck**), if desired, before completing the change to multi-user mode.

The **brc** program performs the tasks of clearing out and reinitializing the /etc/mnttab (file system mount table) file.

The **rc** program establishes the interactive link between between users' terminals and the system and invokes the other programs listed above.

The **cron** program functions as a "clock," allowing a task to be performed at a specified time. The tasks that will be performed are located in a file called "crontab" in the /usr/lib directory.

The **openup** program performs the basic initialization tasks necessary for supporting the serial communication devices such as terminals and serial printers.

The **lpd** program initializes the printer spooler facility in the system. In UNIX System V, there is a choice of two different spoolers which may be invoked. These two spoolers are: **lp** or **lpr**. It is suggested that the **lp** spooler be used because it operates under the printer scheduler (**lpsched**) facility which provides better control and management over the print queue/request mechanism.

Note: the selection of either of the aforementioned spoolers is accomplished by reprogramming the /etc/rc program.

As in the single-user mode, there are certain considerations and precautions the system administrator should be aware of while in the multi-user environment.

One such precaution is to ensure that users do not attempt to alter any system configuration parameter. This can be accomplished by making sure the permissions of files in the root file system are not altered, unless it is necessary.

Another precaution is to avoid leaving the multi-user environment in any way other than what is prescribed in this guide. To do so can leave the system in an unstable state.

Super-user Mode

The **super-user** (su) mode is not actually a separate environment, but rather an unrestricted mode for which one is not constrained by system permissions.

Note: because this is an unrestricted mode, use **extreme caution** while in super-user!

A system administrator can invoke the super-user mode by entering "su" after the normal system prompt character. In addition, super-user mode can be invoked from any directory.

Super-user mode acts as any other login process, asking for a password. After the password has been entered and verified, the system indicates super-user mode by displaying a **pound** sign prompt character on the terminal screen. To leave the super-user mode, simply enter **<control-d>**, and the system will return to the prior login state.

The super-user mode will be used quite often by the system administrator; other users should not have access to the "su" password.

Note: the same password used by "su" is also used for logging into the root account.

Summary

This chapter examines the major features of the UNIX operating system. It is by no means a complete example of the power that is inherent in the UNIX system and the flexibility it offers to the user.

As you become familiar with the UNIX system, you will realize that there are a multitude of unique and useful tools that allow you to be innovative in performing the job of a system administrator.

TABLE OF CONTENTS

CHAPTER 3

CHAPTER 3

INSTALLING A NEW VERSION OF UNIX

Preface

This chapter describes procedures for converting to a new UNIX system.

The "procedures" which follow are conceptually represented, and actual execution of commands is not included here.

Considerations

Each UNIX update package contains a detailed installation document, and the SRN pertaining to the specific computer system involved.

Some of the issues which should be considered before installing a new version of UNIX are:

- System configuration
- System backup
- Compatibility

System Configuration

This applies to maintaining a written record of the system's unique configuration. This record should contain, at the very least, the following:

- The number of file systems created, their names, sizes, and location on the physical disk
- The number of user accounts installed
- Communication "ports" that have been changed to support anything other than a terminal
- Special programs that were incorporated into the "root" directory
- Any changes to the system's features that are allowed to be performed under the UNIX system
- Any other changes pertaining to system configuration that have occurred since original installation
- List of system files that have been modified for this installation

Keeping such a "log" up to date can save a system administrator a significant amount of time and frustration, not only when changing to a new version of UNIX, but also if problems arise in daily operation.

System Backup

The idea of periodically making a copy of the work performed on the system will be iterated throughout this guide. Remember, the loss of important data can be very costly, in terms of both time and money!

Before attempting to install a new UNIX version, make sure there is a copy (backup) of all the users' work **current to that day!** In addition, a copy of the configured version of the "root" file system should already exist. This will be invaluable if reverting to the older UNIX version becomes necessary.

Plan effectively, and inform all users of the coming event so they (users) will know not to perform any major work after the specified period of time.

As with every backup operation performed, the system administrator should verify the file system's integrity **prior** to making a copy.

Compatibility Issue

The issue of "compatibility" means determining whether or not the new version of the UNIX system will have an impact on the normal operation of any special application package such as the **Onyx Office**.

Compatibility information is included in the installation documentation. However, it applies **only** to those special application packages that are a product of or supported by Onyx. If another manufacturer's application package is being used, the system administrator should contact the manufacturer to determine whether or not compatibility is maintained.

Procedures

The following procedure describes in general terms how to make a successful conversion to a new UNIX system.

PROCEDURE: **Converting to a new UNIX system**

1. Plan the transition. Allow enough time for the users to do their part.
2. Read **all** of the installation directions provided **before** attempting to install the new system.
3. If there is some part of the procedure that is unclear, **ask the questions now!**
4. Make the necessary backups of the user data.
5. Install the new system as per the directions provided.
6. Perform any advised integrity checks to ensure the installation is intact.
7. Reconfigure the system to your special requirements.
8. Make a new copy of the reconfigured "root" file system.
9. Update the historical record with any new configuration information.
10. Monitor the system's operation, looking for actions that do not appear normal.

Summary

This chapter discusses the aspects of "system configuration," "system backup," and "compatibility" as they pertain to installing a new UNIX system. In addition, this chapter describes a procedure outlining, in general terms, the steps involved in a new UNIX installation.

It should be noted that such an installation is not an everyday event. However, when necessary, having a basic understanding of the factors involved can avoid confusion, frustration, and delay.

TABLE OF CONTENTS

CHAPTER 4

LIST OF ILLUSTRATIONS

CHAPTER 4

GETTING STARTED

Preface

This chapter defines and illustrates the procedures for setting up and configuring a UNIX system.

Overview

This chapter begins by outlining the way in which a UNIX operating system is transferred from tape to disk, and describes the loading of this operating system into memory. It further describes the creation of file systems, configuring the system, performance considerations, creating user environments, and much more.

Prior to getting started, certain conventions and considerations associated with this chapter should be noted.

The procedures described herein are for reference only! They are provided as a general example of tasks the system administrator may or may not need to perform depending upon the specific Onyx computer system in use.

Command references are in the form of: **name(number)**. Where "name," is the actual command and "(number)" indicates the section in the Enhanced ONYX System V USER, PROGRAMMER, and ADMINISTRATOR REFERENCE MANUAL's where commands are explained in detail.

Remember, **Caution** and **Warning** titles refer to important information that should be noted by the system administrator. Therefore, read all the associated text to these titles carefully!

When references are made to "edit," they mean to alter the contents of the named file with one of the UNIX system editors. As mentioned earlier, there are two primary editors: "vi," and the more primitive "ed." The system administrator should become familiar with the vi editor because of its flexibility.

While studying a procedure, notice that all commands and actions to be performed appear in **bold type**. All system responses to a requested command appear immediately below that command. In addition, all comments made within a procedure are contained between brackets "[]." The symbol <cr> means "carriage return."

Special Keys

There are a number of terminal keyboard control sequences which have either special significance in the UNIX system or are just frequently used. A control sequence means to hold down the <control> key while pressing some other terminal key.

Some of the more commonly used control sequences in the UNIX system include the following:

- Control - d
- Control - h
- Control - x

The **<control-d>** sequence is used to terminate (log off) the computer system or terminate communications while under the control of some of the communications facilities.

The **<control-h>** sequence is used to backspace over previously entered characters. However, without adding some other special parameters, this sequence backs up and writes over the character without erasing it.

The **<control-x>** sequence is used to delete and not execute a command line that has been entered by mistake.

The control sequences of "h" and "x" can be configured in the UNIX system to reflect the terminal's backspace and delete keys respectively.

In a control sequence example, it appears as: **<cntrl- >**, where the actual letter desired will follow to the right of the "-" within the arrows.

Initial Preparation of a UNIX System

The following discussion outlines the two procedures required to be performed in order to prepare, for the first time, a UNIX operating system for operation.

These two procedures include:

- Transferring the "root" file system from tape to disk
- Auto-configuration of device files and the "/usr" file system

Transferring the Root File System from Tape to Disk

This process is performed in the standalone shell environment and involves the loading of the Onyx "Initialization" (INIT) tape which contains the standalone shell programs and the **root** file system.

The example procedure below shows a typical transfer process. However, the system administrator should use the exact procedures described in the SRN that accompanies the system.

Note: during this process, the system administrator should monitor the console display for error messages. If an error occurs during any part of the process, the system administrator should note the error message, proceed no further with the preparation process, and follow the error analysis and recovery procedures described in Chapter 7 of this guide.

PROCEDURE: **How to transfer the root file system**

1. Using the INIT tape, ensure it is in the SAFE position; then insert it into the system.

2. Press the reset button on the system.

   -- PROM 05/03/85-12:19:48 --

   [At this point, the **self test** phase is envoked.]

   0,1,c,9,?:c <cr>

   [Entering "c" instructs the system to load, execute, and boot up from the cartridge tape device. The other options available are: "0," boot from disk drive 0; "1," boot from disk drive 1; "9," boot from nine track tape, and "?" for displaying the definitions of all the options.]

```
SHELL 02/01/85-15:45:12

SHELL$$
```

3. From the console terminal, perform the following:

```
SHELL$$ init  <cr>

Do you want to save a new standalone shell to the disk (y|n)?y <cr>
Saving shell to disk ......................................
Here is the tape directory info block:
ONYX-INIT-TAPE
UNIX 5.1 6810
02/06/85
17:16:14
1
5

The disk on controller 0, unit 0, is already formatted.
Would you like to reformat it (y|n)?n  <cr>
```

[If there are two or more disk drives attached to the system, the program displays the "format" query again. If a drive has been added, respond to the query with a "y"; otherwise, enter "n".]

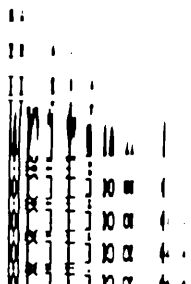
```
The disk on controller 0, unit 0 was already formatted and
configured.
Would you like to reconfigure it (y|n)?n  <cr>
```

[Note: this message reappears if two or more disks exist. **If a disk was reformatted, then it must be reconfigured by entering "y" to the query!** Otherwise, enter "n" as shown above.]

```
Answering "y" to the following question will overwrite (destroy)
what is currently in the root file system ("/") on disk
controller 0, unit 0.

Do you want to load in the new root file system from tape (y|n)?
y  <cr>

Dumping tape to disk.
```



These two procedures include:

- Transferring the "root" file system from tape to disk
- Auto-configuration of device files and the "/usr" file system

Transferring the Root File System from Tape to Disk

This process is performed in the standalone shell environment and involves the loading of the Onyx "Initialization" (INIT) tape which contains the standalone shell programs and the **root** file system.

The example procedure below shows a typical transfer process. However, the system administrator should use the exact procedures described in the SRN that accompanies the system.

Note: during this process, the system administrator should monitor the console display for error messages. If an error occurs during any part of the process, the system administrator should note the error message, proceed no further with the preparation process, and follow the error analysis and recovery procedures described in Chapter 7 of this guide.

PROCEDURE: **How to transfer the root file system**

1. Using the INIT tape, ensure it is in the SAFE position; then insert it into the system.

2. Press the reset button on the system.

   -- PROM 05/03/85-12:19:48 --

   [At this point, the **self test** phase is envoked.]

   0,1,c,9,?:c <cr>

   [Entering "c" instructs the system to load, execute, and boot up from the cartridge tape device. The other options available are: "0," boot from disk drive 0; "1," boot from disk drive 1; "9," boot from nine track tape, and "?" for displaying the definitions of all the options.]

```
SHELL 02/01/85-15:45:12

SHELL$$
```

3. From the console terminal, perform the following:

```
SHELL$$ init  <cr>

Do you want to save a new standalone shell to the disk (y|n)?y <cr>
Saving shell to disk ..........................................
Here is the tape directory info block:
ONYX-INIT-TAPE
UNIX 5.1 6810
02/06/85
17:16:14
1
5

The disk on controller 0, unit 0, is already formatted.
Would you like to reformat it (y|n)?n  <cr>

[ If there are two or more disk drives attached to the
  system, the program displays the "format" query again.
  If a drive has been added, respond to the query with
  a "y"; otherwise, enter "n". ]

The disk on controller 0, unit 0 was already formatted and
configured.
Would you like to reconfigure it (y|n)?n  <cr>

[ Note: this message reappears if two or more disks exist.
  If a disk was reformatted, then it must be reconfigured
  by entering "y" to the query!  Otherwise, enter "n" as
  shown above. ]

Answering "y" to the following question will overwrite (destroy)
what is currently in the root file system ("/") on disk
controller 0, unit 0.

Do you want to load in the new root file system from tape (y|n)?
y  <cr>

Dumping tape to disk.

Writing to sector [sector number]

[ The sector number advances as the root file system data
  is written onto each disk sector. ]

Do you want to save a new standalone shell to the disk (y|n)?n <cr>
Returning to the standalone shell.
SHELL$$
```

4. Remove and store the INIT tape for future use.

Auto-Configuration of Device Files and the "/usr" File System

The second phase of initial UNIX system preparation involves creating the necessary special device files in the "/dev" directory, creating the "/usr" file system, and transferring the /usr file system data from tape to disk.

Note: the auto-configuration phase is performed and encountered **ONLY** after a transfer of the root file system from the INIT tape has been performed! It does not occur upon booting the system for daily use.

PROCEDURE: **How to auto-configure the devices and /usr file system**

1. Press the reset button on the system.

```
-- PROM 05/03/85-12:19:48 --

0,1,c,9,?:0  <cr>

[ "0" instructs the system to boot
  from disk 0. ]

  SHELL 02/01/85-15:45:12

SHELL$$
```

2. From the console terminal, perform the following:

```
SHELL$$ boot  <cr>
Booting /unix.

INIT: SINGLE USER MODE

Completing Initialization procedure.

Creating special files in /dev.

                The scsi config struct

contrl unit 0 unit 1 unit 2 unit 3 unit 4 unit 5 unit 6 unit 7
00000   DISK
00001   CART
00002
00003
00004
00005
00006
00007
```

```
[ Display indicates all currently configured and
  physically existing devices. ]

Aswering yes to the following question will overwrite
(destroy) any information on slice 3 (/dev/dsk/c0d0s3).

Do you want to create "/usr"? y  <cr>
Creating a file system on slice 3 for "/usr"...

bytes per logical block = 1024
total logical blocks = 22374
total inodes = 5584
gap (physical blocks) = 1
cylinder size (physical blocks) = 1

[ Total number of logical blocks and i-nodes configured
  depends upon the actual storage capacity of the disk. ]

[ Next phase labels the new file system. ]

Current fsname: , Current volname: , Blocks = 44744,
Inodes = 5552, FS Units: 1Kb, Date last mounted:[date]
NEW fsname = usr, NEW volname = S3 -- DEL if wrong!!

[ Next phase indicates the disk storage used by the
  configured file systems. ]

/usr  (/dev/dsk/c0d0s3): 22022 blocks 5550 i-nodes
/     (/dev/dsk/c0d0s1):  4527 blocks 1670 i-nodes

Please insert the release CPIO tape for "/usr" and press return.

<cr>

Reading from tape...

[ System transfers all /usr files from tape to disk. ]

                    10700 blocks
DONE.

Initialization Complete.

Going to INIT state 2 (Multi-user)

INIT: New run level 2

ONYX onyx 5.1 6810

Do you want to check the file systems (y or n)? y <cr>
```

```
/dev/dsk/c0d0s1
file system: root  Volume: [name/number]
** Phase 1 - Check Blocks and sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Free List
[N] files  [N] blocks   [N] free

/dev/dsk/c0d0s3
file system: usr  Volume: S3
** Phase 1 - Check Blocks and sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Free List
[N] files  [N] blocks   [N] free

[ "N" indicates the number of files and blocks
  configured which may vary among systems. ]

Current date: [date]
Error logging started.
Cron started.

Mounted file systems -

/usr (/dev/dsk/c0d0s3): 16471 blocks  5073 i-nodes
/    (/dev/dsk/c0d0s1):  4529 blocks  1672 i-nodes


********* SYSTEM MULTI-USER [date] *********

Console login:
```

4. Remove and store the /usr tape for future use.

```
********************** CAUTION **********************************
*                                                                 *
* Other tasks may need to be performed to complete the            *
* initial configuration of the system such as establishing        *
* user accounts or altering the system's default parameters       *
* prior to allowing users to start their work.  Performing        *
* such tasks requires the system to be in the single-user         *
* environment; therefore, it may be necessary to execute          *
* the "shutdown" command to effect the mode change.               *
*                                                                 *
*******************************************************************
```

Booting Up the System for Normal Use

The "bootup" process is generally performed in three phases: first, the system "self test" phase; second, loading the necessary data from disk to system memory; and third, transferring control to the kernel.

Note: initialization default state (mode) is set to "2."

The **self test** phase performs an array of "go-nogo" tests that ensure system hardware integrity. This array may include testing the system memory elements, the serial communications ports, and so on. A failure at this level means the system cannot function enough to achieve an operational state.

The **loading** phase, or "bootup," involves reading the bootup data from disk and loading this data into system memory to be acted upon.

The final phase, **multi-user**, is achieved as a result of the successful execution of the bootup data. It is, as already described, the normal operational environment used under the UNIX system.

The following procedure describes how to boot up the system.

PROCEDURE: **How to boot up the system for normal use**

1. Press the system reset button. The console terminal should display the following:

   -- PROM 05/03/85-12:19:48 --

   0,1,c,9,?: 0 <cr>

   SHELL 02-01-85-15:45:12

   SHELL$$

   [Note: approximately one minute elapses between the "PROM" and "SHELL" messages. This time is used for running the self test programs. ' Unless an error occurs, no messages are displayed during the self test phase.]

2. Perform the following:

   SHELL$$ **boot** <cr>

   Booting /unix.

   Console login:

Checking the Basic File System

The methods used in the UNIX system to verify the integrity of a file system are the **fsck** program or **dfsck** if there is more than one disk drive configured.

The **fsck(1M)** program checks a file system based upon five criteria.

They are:

- Verifies proper blocks and their sizes
- Verifies the pathname structure
- Verifies the connectivity of i-node pointers
- Verifies the reference count of the i-node list
- Verifies the list of free blocks

In addition to these checks, the program displays the total number of files listed, the number of disk blocks used by these files, and the number of disk blocks still available for use.

When **fsck** is invoked, it checks all of the file systems existing on the system. **Fsck** gets its information by reading the "checklist" file in the /etc directory. Checklist contains the names of all the file systems and should be updated each time one is added.

The following procedure illustrates the use of **fsck.**

Note: do not attempt to check a mounted file system! If the system is currently in multi-user, unmount the desired file system using the **umount(1)** command or revert to single-user mode.

PROCEDURE: **How to use the fsck(1M) command**

1. Perform the following.

```
# fsck  <cr>

/dev/dsk/c0d0s1
file system: [name]  Volume: [name/number]
** Phase 1 - Check Blocks and sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Free List
[N] files  [N] blocks   [N] free
#
```

The feature content and diagnostic messages inherent to fsck are quite extensive. Diagnostic information is contained in Chapter 7. Feature content is contained in the FSCK(1M) section of the Enhanced ONYX System V ADMINISTRATOR REFERENCE MANUAL.

Configuring the Operating System

This section describes how to configure the operating system.

Overview

The UNIX system, as installed from the Onyx INIT tape, establishes basic default configurations for:

- Pre-mapped file system boundaries
- Basic number of users supported
- Basic support for serial and parallel printers

With everyday use, some of these configuration parameters may need to be altered, or some unused facility may need to be activated.

System Default Configurations

The basic system default conditions set by the Onyx INIT tape include the following:

- The system is configured for at least six terminals, including console.
- The system establishes the terminal baud rate at 9600 baud.
- The system is configured for a parallel printer.
- The system is configured for a serial printer at 9600 baud.
- The system has pre-mapped (sliced) disk storage.
- The time, date, and timezone (TZ) parameters have not been established.
- The Bourne (sh) shell is activated upon logging in.
- A basic "profile" file has been created.
- The prompt characters **SHELL$$** - standalone, **#** - root or super-user, and **$** - users, have been set.

- The login accounts - root, uucp, bin, adm, and sync are established but without password protection.
- In multi-user, the super-block is updated every 30 seconds.

UNIX Kernel Sizing and Parameter Selection

This section describes those UNIX kernel parameters that are altered to achieve optimal operating system performance or accommodate some special operating system environment.

Note: **Onyx has already configured the kernel parameters for optimal performance!** Making changes to these parameters requires an in-depth knowledge of the UNIX system; therefore, **DO NOT** alter any kernel parameter arbitrarily!

The major parameters of concern are found in the system "description" file - dfile "configuration" file which resides in the /src/uts/cf directory.

The dfile File

The contents of dfile may appear as follows:

```
*          (#)dfile.m4  2.1
*
* Motorola VME/10 Development System
*
* dev    vector   addr     bus      count
*
disk     3d0      0        5
clock    130      0        6        1
cart     3d4      0        5
dman     190      0        7        1
scsi     78       0        7        1
con      100      0        5
con      104      0        5
con      108      0        5
lptr     180      0        5        1
dmae     194      0        7        1
*
* Traps - Not to be modified
*
buserr   8        0        7
*
* System Devices
*
root     disk     1
swap     disk     2        1        3000
pipe     disk     1
dump     disk     0
*
* Tunable Parameters
*
buffers            50
calls              50
flocks             200
inodes             90
files              60
mounts              8
procs              50
texts              40
clists             150
sabufs              0
maxproc            25
swapmap            75
hashbuf            64
physbuf             4
power               0
mesg                1
sema                1
shmem               1
```

The first section denotes all the hardware devices acknowledged by the system. In addition, this section lists the interrupt **vector** addresses, link (**addr**) addresses, **bus** assignments, and the device **count** associated with these devices.

Note: where the **count** is unspecified, the default of "1" is assumed.

The second section denotes the **vector** address, link (**addr**) address, and **bus** assignment for communicating error conditions on the bus.

The third section denotes major system disk devices currently configured on the system. These devices are described briefly as follows.

The **root** mnemonic specifies the device where the root file system is located. Normally, **root** is logical disk drive 1, section 1.

The **pipe** mnemonic specifies where "pipes," the function of using the **output** of a command as the **input** to another command, are allocated within a mounted file system (normally the root).

The **dump** mnemonic specifies the device used by the system for storing the contents of the system memory after a crash. Currently, this information gets dumped into /dev/dsk/c0d0s0.

The **swap** mnemonic refers to the device and how many disk blocks are allocated for swapping. In the example above, **swap** is configured such that file system "c0d0s2" is the allocated storage area for swapped data, "1" indicates the first starting block ("swplo" value) of the storage area, and "3000" (the "nswap" value) signifies the total number of blocks usable.

Note: "swplo" and "nswap" are parameters defined in the **conf.c** file.

The last section denotes the major tunable parameters and their values currently configured. These parameters are discussed later in this section.

To display a "snapshot" of all the tunable parameters currently configured, the system administrator uses the **sysdef(1M)** command as shown in the following procedure.

PROCEDURE: How to use the sysdef program

1. Perform the following:

```
# /etc/sysdef   <cr>

UNIX SYSDEF: version= unix5.1, node= rel
System Size: I=51214, D=35110

Rootdev: Maj. 0, Min. 1
Swapdev: Maj. 0, Min. 2
Pipedev: Maj. 0, Min. 1
*
* Tunable Parameters
*
swplo   1
nswap   3000
buffers 50
sabufs  0
procs   50
mounts  8
inodes  90
files   60
calls   50
texts   40
clists  150
swapmap 50
hashbuf 64
physbuf  4
maxproc 25
maxlock 200
power   0
mesg    1
sema    1
shmem   1
#
```

The following describes briefly the function of each tunable parameter.

```
swplo    -- swap starting block
nswap    -- number of swap blocks
buffers  -- number of buffers
procs    -- number of processes
mounts   -- number of mounted file systems
inodes   -- number of open i-nodes
files    -- number of open files
calls    -- number of time invoked functions
text     -- number of active text segments
clists   -- number of character list buffers
swapmap  -- size of free swap block list
hashbuf  -- number of hash buffers
physbuf  -- number of header buffers
maxproc  -- maximum number of user processes
maxlock  -- maximum number of locked files
power    -- powerfail condition code
mesg     -- enables message buffer parameters
sema     -- enables semaphore parameters
shmem    -- enables shared memory parameters
```

The following text briefly defines the pertinent parameters which may be altered.

The **swplo** parameter, as described earlier, represents the starting block location in the allocated "swap" disk storage space.

The **nswap** parameter, also described earlier, defines the size or number of blocks used by the "swap" disk storage space.

The **buffers** parameter determines the number of system buffers to allocate. It can be noted that in a "real-time" operating environment, increasing the number of available buffers can improve system response.

The **procs** parameter determines the number of entries to be allocated for the process table. Each entry represents an active process. Within the table, the first and second entries are always the "scheduler" and "init" processes, respectively. The number of entries depends upon the number of terminal lines available for use and the number of processes created by each active user. Each user logged onto the system generates an average of from two to five processes.

When the table is full or overflows, the system issues error messages such as "no more processes" or "EAGAIN."

The **mounts** parameter specifies the number of entries allocated for the mount table. Each entry represents a mounted file system. The **root** file system is always the first entry. When the table is full, the system issues a "BUSY" error message.

The **inodes** parameter determines the number of entries allocated for the i-node table. Each entry represents a unique available i-node. When the table overflows, the system issues an "Inode table overflow" error message. If this error message is experienced regularly, the size of the table is too small and should be increased. The number of entries used depends upon the number of processes, text segments, and mounted file systems currently active on the system.

The **files** parameter specifies the number of file table entries to allocate. Each entry represents an open file. When the table overflows, the error message: "no file" is issued to the console terminal. If this error is experienced often, the table size should be increased.

The **calls** parameter determines how many entries are allocated for the call-out table. Each entry represents a function to be invoked at a later time by the clock handler. The unit of time used is 1/60 of a second. The call-out table is used by the terminal handler program, and various other I/O handler programs, to provide a delay in I/O interaction. When the table overflows, the system halts and issues the message: "Timeout table overflow."

Note: the value of **calls** must be greater than two (2)!

The **texts** parameter specifies how many text table entries to allocate. Each entry represents an active read-only text segment. Such a segment is created by using the "-i" or "-n" options of the **ld(1)** command. When the table overflows, the system issues an "out of text" error message.

The **clists** parameter determines the number of character list buffers to allocate. Each buffer contains up to 64 bytes. The buffers are dynamically linked together to form **input** and **output** queues for the terminal lines and various other slow speed I/O devices. The average number of buffers needed for each terminal line ranges from five to ten. When the buffers are full, any further characters queued are lost.

The **swapmap** parameter determines the number of entries to allocate for the free list of swap blocks. It represents free blocks in the swap area in units of 1024 byte blocks.

The **maxproc** parameter specifies how many concurrent processes a nonsuper-user can invoke at any given time.

Guidelines for Kernel Sizing and Altering Parameters

The following text examines the considerations involved with altering parameters and sizing the kernel, and offers some suggestions for determining their proper sizes and values.

In many cases there is a tradeoff between getting all of the desired resources into the system and keeping the system from growing too large. One way to reduce the size of the kernel is to leave out or remove the device driver software for non-essential devices. Doing so may decrease the overall kernel size by a few hundred to over a thousand bytes! However, most systems do not have "extra" devices which can be left out or removed. Therefore, the only way to trim the kernel size is to reduce the allocation of some of the system's resources.

To aid in the decision of which resources to reduce, each pertinent system parameter is discussed in further detail below.

Buffers -- buffers

In the UNIX system, there is a "cache" or group of buffers which functions transparently to the user and through which most disk I/O is funneled. The system attempts to maintain in memory copies of those blocks of disk data that are most often accessed, thereby reducing the number of I/O operations needed to access those blocks. As the size of the buffer cache is reduced (**buffers**), the length of time a particular block remains in memory is reduced. This results in a reduction in the effectiveness of the caching and an increase in the amount of disk I/O operations that must be performed. Concurrently, when the cache size is insufficient to accommodate the number of processes allowed, system performance suffers dramatically.

When the cache buffering is properly sized, there are enough available buffers to accommodate those processes which require them and allows processes to perform in parallel rather than waiting for a buffer to free up.

The size of the buffer cache (**buffers**) should never be less than the number of active processes allowed! To estimate the number of active processes, use the formula below.

```
# of active processes = (# of concurrent users) + (.15 * procs)
```

A system can function with as few as 10 or 15 buffers, but 30 is the recommended minimum. Each buffer added increases the size of the kernel by approximately 540 bytes. Conversely, deleting a buffer decreases the kernel by the same amount. Adding buffers visibly improves system performance until the number of available buffers exceeds the number of active processes by a factor of three (3). Beyond that point, no significant gain in performance is achieved.

Processes -- **procs**

There is a firm upper limit in every UNIX kernel on the number of concurrent processes (active and inactive) that are supported. Reaching or exceeding this limit causes invoked processes to fail or wait until the system can accommodate the process. If this condition arises often, it can be alleviated by raising the **procs** limit. However, each increase to the limit adds approximately 40 bytes to the size of the kernel.

To approximate the "minimum" value for **procs**, the formula is as follows:

```
procs = (# of terminals) + (1.5 * (concurrent users + 10))
```

Where:

- 1.5 -- accounts for the required process associated with every terminal exclusive of other processes.
- 10 -- accounts for system processes such as daemons that are required for normal operation.

The recommended **lower** limit is 40 or 50; the recommended **upper** limit should not exceed 200.

Mounts -- **mounts**

Each UNIX kernel has a configured limit on the number of file systems that can be mounted at the same time. Once this limit is reached, no additional file systems can be used until one of the file systems currently in use is unmounted. Increasing the size of the mount table adds approximately 12 bytes to the kernel size. In addition, each mounted file system consumes one of the system buffers while it is in a mount state.

In determining the mounts value to set, calculate the number of file systems needed for the installation and then increase that number by one or two. The recommended minimum number of mountable file systems is four or five.

I-nodes -- inodes

Each UNIX kernel has a configured limit on the number of files that can be concurrently in use. Once this limit is reached, no additional files can be used and system activity is impeded until some of the active processes have relinquished references to some of the open files.

Conceptually, each process uses approximately 20 separate files. However, this condition seldom occurs; therefore, allocating two or three i-nodes for each process is quite adequate. Increasing the number of i-nodes the system uses adds 32 bytes to the kernel size for each additional i-node. The recommended minimum number for inodes ranges from 90 to 100.

Files -- files

The distinction between an i-node and a file is subtle. As described earlier, each file has an associated i-node. Each open file on the system has a file table entry. If the table entries are exhausted, no new files can be opened until others have been closed. Increasing the number of entries in the file table adds eight bytes to the kernel size for each additional entry. Again, the recommended minimum number of entries is from 90 to 100.

Calls -- calls

The UNIX system supports internal, time-based calls; meaning certain functions are scheduled for action at a specific time. These calls are typically used for time related activites within a device driver's software. Exhausting the number of entries in the call table causes the system to halt operation! Therefore, it is prudent to ensure the parameter is large enough to prevent this condition. To determine a safe lower limit, use the following formula:

calls = (.25 * (# of terminals)) + (# of terminals)

The recommended minimum number of entries is 30 and each added entry increases the size of the kernel by eight bytes.

Texts -- texts

The UNIX system supports shared load modules; one copy that is shared among all users who are executing the same program. This feature contributes enormously to system efficiency when programs such as the shell, editors, and the C compiler are being constantly invoked by many users. To provide this feature, the

system must maintain a special table containing an entry for each actively shared text segment. If the maximum number of entries is reached, no additional shared text programs are allowed until some of the processes involved have completed.

The addition of an entry to the table increases the size of the kernel by approximately 16 bytes. A reasonable estimation of the minimum number of shared text segments required is 30% of the number of processes on the system. The recommended minimum value for **texts** is from 25 to 30.

Swapmap -- **swapmap**

UNIX manages its memory and swap space through the use of variable size maps. These maps grow as the demand for space increases; however, when either table reaches its maximum limit, the system becomes very inefficient. A reasonable estimate for the size of these parameters is 1/3 to 1/2 the number of processes on the system. The recommended minimum value to set for both tables is 40 or 50.

Tty Buffering -- **clists**

There are several parameters that control the buffering of terminal input and output (I/O). These parameters have a significant effect on system throughput and on what the user sees.

The first set of parameters involved is: **tthiwat** and **ttlowat**, the high and low water marks for buffering. These parameters are found in the tty.h file in the /usr/include/sys directory. The **tthiwat** value sets the maximum number of output characters that is buffered for a terminal before the writing process is blocked to await a "draining" of the queued backlog. The **ttlowat** value establishes the level at which the writing process is allowed to continue outputting characters to the terminal.

Therefore, the larger the value of **tthiwat**, the more output characters a process is allowed to queue before suspension. If the value is small, a process is suspended many times while attempting to output a large number of characters. This suspended action increases process overhead in terms of additional context switching and device interrupt generation, thereby decreasing system throughput.

The **ttlowat** value also affects how the outputted data is displayed on the terminal. When the output queue has been drained to the low water mark, the suspended writing process is awakened. However, even after the process is awake, it may not run immediately. If the value of **ttlowat** is high enough to prevent draining the queue before the process runs again, the output on the terminal will appear continuous.

Conversely, if the value is too small the output data appears "choppy" due to pauses between the time that enqueued output completes and the writing process starts again.

When very low speed terminals (300 baud or below) are used, there is no reason to buffer large amounts of data; at 300 baud, only 30 characters are outputted a second. Therefore, if **tthiwat** is set to 60 and **ttlowat** to 15, the buffer would enqueue two seconds worth of characters (60) before the writing process was suspended. After the process was awakened, it would have a full half-second to enqueue more output. Under this circumstance, the output would appear continuous.

However, using these same values (60 and 15) on a terminal line at 9600 baud would produce a "choppy" output because of an insufficient time interval between outputting the enqueued data and restarting the writing process. The time interval involved in this case equates to 60 milliseconds for flushing the buffer and 15 milliseconds to restart the writing process. For systems using terminals at low speeds, the values for **tthiwat** and **ttlowat** described above are adequate. For systems using higher speed terminals (9600 baud is common), values such as 200 for **tthiwat** and 70 for **ttlowat** are recommended.

Once the high and low water mark values have been established, an adequate number of buffers must be allocated to hold the enqueued output and terminal input characters. The parameter which specifies the number of buffers called "clists" is **clists**.

Note: the **slists** value represents the amount of buffering for all terminals combined!

Each **clist** buffer holds up to 64 bytes and the segments are maintained outside of the kernel's address space; therefore, **clists** do not increase the size of the kernel. However, **clists** do reduce the amount of memory space available for use by user processes. In determining how many **clists** to allocate, the value set must be adequate to handle all the necessary buffer demands. At best, there should be enough buffers to accommodate a full input line (80 characters) and a full output queue (**tthiwat** characters) for every terminal used on the system.

However, as stated earlier, each **clist** consumes valuable memory space for user processes and it is unlikely that all the system terminals would fill all of the buffers at the same time. Therefore, a safe number of **clist** buffers to allocate can be determined using the following formulas:

clists = (# of characters)/6

Where:

# of characters = (.65 * (# of terminals)) * (tthiwat + 65)

Considerations for Testing a Modified System

Once the parameter changes have been made and a new kernel generated, the system administrator should test the new kernel before it is installed as the standard system.

Testing the new kernel involves booting it up and operating as normal while monitoring system activity for any adverse effects. If any adverse conditions occur, make note of the symptoms exhibited by the system, isolate the problem, and effect a repair. If the problem cannot be readily determined, reinstate operation by rebooting the old kernel and then recheck all the changes made to the modified system.

```
****************** WARNING ************************
*                                                   *
* If changes were made to the allocation of         *
* the root or swap disk area, ensure the values     *
* selected DO NOT CAUSE AN OVERLAP ONTO ANOTHER     *
* ALLOCATED DISK SPACE!                             *
*                                                   *
*****************************************************
```

Making the desired changes involves using one of the system editors and generating a new kernel under a different name for testing. The procedures for generating and booting an alternate kernel are described in this chapter under the sub-heading of "Rebuilding the Operating System."

Creating File Systems

As discussed in Chapter 2, Onyx initially configures four file systems on the computer system. The last of these four file systems is **usr**. The **usr** file system boundaries start at the ending boundary of the **swap** file system and continues to the end of the disk storage. The beginning portion of the **usr** file system contains all the additional user utilities and their libraries leaving the rest of the file system storage available for users' work. The actual size of the users' workspace depends upon the physical storage capacity of the disk drive in use; however, the workspace allocated in **usr** is generally adequate for most installations.

```
************************** CAUTION ************************************
*                                                                     *
* If the particular needs of a given installation dictate that        *
* the usr file system be partitioned further into additional          *
* file systems, then this decision should be made prior to allowing   *
* users to start their work in the usr file system!                   *
*                                                                     *
***********************************************************************
```

The reason for this is that creating additional file systems on a disk requires repositioning the file system boundaries and reconfiguring the disk. Since a file system's data cannot be truncated into another file system, repositioning the boundaries after data has been written into the file system could result in the loss of that portion of the data where the new boundary would exist.

Planning and determining the particular needs of the installation is one of the important jobs of the system administrator!

To make an additional file system requires performing the following steps:

1. Determining the sizes of the file systems currently configured.
2. Reconfiguring the disk storage boundaries and sizes.
3. Creating the additional device "nodes."
4. Making the additional file system.
5. Labeling the additional file system.

Determining the Sizes of the File Systems

The Onyx configured file systems and their sizes are defined as follows:

c0d0s0 -- reserved area, 1188 blocks in size

c0d0s1 -- root file system, 8000 blocks in size

c0d0s2 -- swap file system, 3000 blocks in size

c0d0s3 -- usr file system, 4311 blocks used and the remaining disk storage allocated for workspace

Reconfiguring the Disk Storage Boundaries and Sizes

The following example procedures are based upon creating and configuring an additional file system whose directory name is **wrkspc**, device name is **c0d0s4**, and block size as starting from the end boundary of the **usr** file system and continuing to the end of the available disk storage.

The reconfiguration process is accomplished by using the **diskconf** program -- one of the utilities contained in the standalone shell.

Note: this procedure requires the system to be reset in order to achieve the standalone shell environment! In addition, the **diskconf** program destroys the spare sector information residing on track 0; therefore, perform the **sparelist** program first and write down the sectors currently spared.

PROCEDURE: **How to use the diskconf program**

1. From the standalone shell, display the current spare sector data as follows:

   SHELL$$ **sparelist /dev/dsk/c0d0s0** <cr>

   Spare information for Controller 0 Unit 0 slice 0

   Base sector [number]
   Number of sectors [number]
   Number of Alternates [number]
   Number of bad sectors [number]

   [If sectors have been spared previously, the following additional data will appear:]

   Sector [number] -> Physical [number]

   [Otherwise, the program displays...]

   There are no spared sectors on this slice.

   SHELL$$

2. Write down all spared sector information.

3. Configure the disk as follows:

   SHELL$$ **diskconf -c 0 -d 0 -s 1188 8000 3000 4000 0 -b** <cr>
   Loading /stand/diskconf.

   [Where:
   -c 0 = the controller bus address.
   -d 0 = the physical drive number.
   -s 1188 = the size of c0d0s0
   8000 = the size of c0d0s1
   3000 = the size of c0d0s2
   4000 = the size of c0d0s3
   0 = the default instruction to make c0d0s4 the rest of the available storage
   -b = the "boot" flag which configures the disk for booting.

```
diskconf: new slice structure - number of slices 5
    slice 0, offset 0, size 1188
    slice 1, offset 1188, size 8000
    slice 2, offset 9188, size 3000
    slice 3, offest 12188, size 4000
    slice 4, offset 16188, size 18372
    slice 5, offset 0, size 0
    slice 6, offset 0, size 0
    slice 7, offset 0, size 0

[ This configuration establishes file system
  c0d0s4 as being 18372 blocks in size. ]

SHELL$$
```

Creating the Additional Device Nodes

This task is accomplished by using the **devices** utility which resides in the /etc directory. **Devices**, when invoked, creates all the additional disk drive device files (nodes) to support eight file systems automatically. In addition, **devices** configures these nodes for both the "blocked" and "character" device modes. The list of blocked disk devices resides in the /dev/dsk directory and the character disk devices in the /dev/rdsk directory.

Note: the **devices** program must be performed in the single-user environment and the actual "device" (e.g., disk, cartridge or nine-track tape) must be turned on! In addition, the "-w" option means "write" or create the entries.

PROCEDURE: **How to use the devices program**

1. In single-user mode, perform the following:

   ```
   # /etc/devices -w  <cr>
   #
   ```

2. Verify the nodes exist in the /dev/dsk and /dev/rdsk directories for the new file system.

   ```
   # ls -l /dev/dsk/c0d0s4  /dev/rdsk/c0d0s4 <cr>

   br--------- 1 root system 0, 4 [date/time]/dev/dsk/c0d0s4
   cr--------- 1 root system 4, 4 [date/time]/dev/rdsk/c0d0s4
   #
   ```

Making the Additional File System

The additional file system is created using the **mkfs(1)** utility as illustrated in the following procedure.

Note: the **mkfs** program must be performed in the single-user environment!

PROCEDURE: **How to make the file system**

1. Make the file system.

```
# mkfs /dev/dsk/c0d0s4  <cr>

bytes per logical block = 1024
total logical blocks = 18374
total inodes = 4592
gap (physical blocks) = 1
cylinder size (physical blocks) = 1
#
```

Labeling the Additional File System

The last step involved in this process is labelling the newly created file system. This is done using the **labelit(1)** command as shown in the following procedure.

Note: the **labelit** program must be performed in the single-user environment!

PROCEDURE: **How to use the labelit command**

1. To label the file system, perform the following:

```
# labelit /dev/dsk/c0d0s4 wrkspc DRIVE0  <cr>

Current fsname:[blank], Current volname:[blank], Blocks: 36744,
Inodes: 4560, FS Unit: 1Kb, Date last mounted:[date],
NEW fsname = wrkspc, NEW volname = DRIVE0 -- DEL if wrong !!
#
```

Upon successful completion of these steps, the system administrator should verify the newly created file system's integrity by performing the **fsck(1M)** program before advancing to multi-user mode.

Installing Optional Device Drivers

This section describes briefly the installation of a new or different device "driver."

The addition of a new driver is not an everyday occurrence. However, when it is necessary to do so, the new driver software is supplied on tape, and a complete set of installation instructions accompanies this tape.

The new device driver is typically in the form of a binary object file suffixed by a ".o".

The directory where the file will reside varies. However, the files that have to be manipulated to incorporate a new driver are fairly consistent. These files are "unix" -- the kernel, "dfile" -- a system device table, the special device file which is generated and placed in /dev, and the appropriate device driver entry for the /etc/master file.

The upper entries appearing in /etc/master define certain characteristics about the device as shown in the example below.

```
*       (#)master.m4  2.1
*
* The following devices are those that can be specified in the system
* description file.  The name specified must agree with the name shown.
*
clock   4   0   302   clk   0   0   0   1   6
con     4   137 104   scc   0   0   0   32  5
lptr    4   33  104   lp    0   0   6   1   5
disk    4   36  154   dsc   0   0   4   32  5
cart    4   37  154   ct    0   1   5   32  5
scsi    4   0   102   sc    0   0   0   1   7
dman    4   0   102   dma   0   0   0   1   7
dmae    4   0   102   dmae  0   0   0   1   7
```

Note: determining these device driver characteristics requires access to the UNIX source code and an understanding of the internal structure of UNIX.

The following procedure describes the general tasks a system administrator performs in incorporating a new device driver into the UNIX system. During such an installation, the system should be in single-user mode with the **/usr** file system (c0d0s3) mounted manually.

PROCEDURE: **How to install a new device driver**

1. Change to the major directory where the driver will reside. For example, /usr/lib.

   ```
   # cd /usr/lib <cr>
   #
   ```

2. Make a sub-directory to contain the contents of this tape. For example, "newdrvr."

   ```
   # mkdir newdrvr <cr>
   #
   ```

3. Change directories to newdrvr.

```
# cd newdrvr <cr>
#
```

4. Insert the tape into the system, and read it into this directory.

```
# cpio -ivBdu < /dev/rmt_rwd  <cr>

/[directories]
 /[files]
[total number of] blocks
#
```

[Note: the "rmt_rwd" tape device is used as an example. The tape containing the needed file should specify what track the file is actually on.]

5. Change directories to /dev, and make a new special device file. For example, "driver."

```
# cd /dev <cr>
#

# mknod /dev/driver [type][major][minor] <cr>
#
```

6. Change directories to /etc.

```
# cd /etc  <cr>
#
```

7. Edit dfile and add the driver name "driver" to the list.

```
*
* Motorola VME/10 Development System
*
* dev    vector   addr   bus   count
*
disk     3d0      0      5
clock    130      0      6     1
cart     3d4      0      5
dman     190      0      7     1
scsi     78       0      7     1
con      100      0      5
con      104      0      5
con      108      0      5
lptr     180      0      5     1
dmae     194      0      7     1
driver   198      0      5     1
```

8. When done, write and quit the file.

9. Reconfigure the dfile.

```
# config dfile <cr>
#
```

10. Edit /etc/master and add the device driver entry.

11. Change directories to /usr/src/uts.

12. Make a new version of the kernel using the **make(1)** command. For example, "new."

```
# make  <cr>

[ The system responds with a variety
  of messages. ]
#
```

13. Rename the new kernel.

```
# mv unix unix.nu  <cr>
#
```

14. Move the newly created kernel version to the root directory.

```
# mv unix.nu  /   <cr>
#
```

15. Change directories to root (/).

16. Save the old version of the kernel as follows:

```
# mv /unix /unix.old <cr>
#
```

17. Make the new kernel the one that will be booted upon initializing the system.

```
# mv /unix.nu /unix <cr>
#
```

18. Update the disk super-block.

```
# sync;sync  <cr>
#
```

19. Unmount the /usr file system.

```
# umount /dev/dsk/c0d0s3  <cr>
#
```

20. Press the reset button and boot the system as usual.

Setting Up Asynchronous Communications

This section describes the most common configuration functions performed by the system administrator pertaining to communications.

These configuration functions include:

- Configuring and installing terminal ports
- Configuring the system to use modems
- Configuring the system for **cu**
- Configuring the system for **uucp**

Configuring and Installing Terminal Ports

This procedure includes the construction or modification of an existing terminal port, and may involve adding or altering parameters within the following system files: /etc/inittab and /dev.

The contents of /etc/inittab determine certain operational characteristics about a serial port such as its environment, its name, the terminal mode, its speed and any special allowable features.

For example:

```
03:2:respawn:/etc/getty tty03 9600155
```

Where:

```
      03 = the "id" or actual tty port number.
       2 = the "rstate" such as "5" for
           single-user or "2" for multi-user
           mode.
 respawn = the "action" or initialization and
           process control setting for the port.
/etc/getty = the init program used for terminal mode
           setup.
   tty03 = the specific /dev device name file.
9600155 = the speed or "baud rate" setting.
```

Note: there are many other "action" settings available for port control. A detailed description of all the available settings is shown in the Enhanced ONYX System V PROGRAMMER REFERENCE MANUAL under the sub-topic of **inittab(4)**.

Onyx offers two options in regards to data output control of a serial device. These two serial output control options are:

- Direct memory access control
- Non-direct memory access control

Direct memory access (DMA) control, as it applies to serial ports, means transferring data in large blocks to a terminal or other serial device. Having this ability increases significantly the speed at which data is presented to and displayed on a terminal, or other serial device. In installations where constant retrieval and presentation of data occurs, DMA output control becomes a valued asset over the non-DMA output (character-by-character) method. The primary reason that non-DMA output control exists in the system is that the console terminal port does not support DMA output control. Therefore, the system administrator should take advantage of this Onyx feature whenever possible. For convenience, Onyx has already configured the appropriate serial ports for DMA output control.

The directories, "/dev/tty_dma" and "/dev/tty_ndma" contain the terminal (tty) port device entries for DMA and non-DMA support, respectively. If the entries for "tty03" in both directories were listed using the **ls -l** command, they would appear as:

```
tty_dma:

crw--w--w- 2 root system 0, 131 [date/time] tty03

tty_ndma:

crw--w--w- 1 root system 0, 3 [date/time] tty03
```

The major differences in the two listings are the **link count** and the **minor device number**. For non-DMA control, the link count is "1," unless altered manually by the administrator; the link count for DMA control is "2," meaning that it has been logically linked to the actual tty03 output device file for normal use. The minor device number listed for the non-DMA control is "3," reflecting the actual tty port number; the minor device number listed for DMA control is "131." This minor device number was derived by adding "128" to the actual tty port number thereby differentiating DMA from non-DMA devices.

The following procedure outlines the configuration of a system to support eight additional ports numbered tty06 to tty13.

Note: this procedure must be performed in the single-user mode. If the system is currently in multi-use mode, use the **/etc/shutdown** command to bring the system to single-user mode.

PROCEDURE: **How to configure and install additional tty ports**

1. Ensure the system is in single-user mode by performing the following:

```
# init s  <cr>

INIT: New run level: S

INIT: SINGLE USER MODE

Current date:[date]
#
```

2. Mount the /usr file system as follows:

```
# mount /dev/dsk/c0d0s3 /usr  <cr>
#
```

3. Change directories to /usr/src/uts as follows:

```
# cd /usr/src/uts  <cr>
#
```

4. Make the new kernel as follows:

```
# make tty14  <cr>

   touch cf/dfile14
   cd cf; make "DFILE=dfile14"
   /etc/config dfile14
   cc -c -O -I/usr/include conf.c
   ar rv ../cflib.a conf.o
r - conf.o
   rm -f conf.o
   /lib/cpp -P -DASM  -I/usr/include m68kvec.s>m68kvec.i
   as -o m68kvec.o m68kvec.i
   ar rv ../cflib.a m68kvec.o
r - m68kvec.o
   /lib/cpp -P -DASM -I/usr/include low.s >low.i
   as -o low.o low.i
   ar rv ../cflib.a low.o
r - low.o
   rm -f name.o
   ../cflib.a is ready
   ld -e start -o unix cflib.a iolib.a mllib.a iolib.a
   oslib.a cflib.a iolib.a mllib.a oslib.a
   unix is ready
#
```

5. Rename the new kernel as follows:

```
# mv unix unix.nu <cr>
#
```

6. Change directories to /dev.

```
# cd /dev   <cr>
#
```

7. Make the special device files using the **mknod(1)** command as follows:

```
# mknod tty06 c 0 134  <cr>
#
```

[Repeat the command line for each tty up to and including tty13. Remember, add 128 to the tty number to get the proper minor device number.]

8. Link the new tty device files for DMA output control as follows:

```
# ln /dev/tty_dma/tty06 /dev/tty06  <cr>
#
```

[Repeat the command line for each tty up to and including tty13.]

9. Change directories to /etc.

```
# cd /etc  <cr>
#
```

10. Edit the inittab file and add the new tty entries after tty05.

```
05:2:respawn:/etc/getty tty05 9600155
06:2:respawn:/etc/getty tty06 9600155
```

[Repeat the task, making the necessary changes, until all tty's are entered.]

11. When done, write and quit the file.

12. Change directories to root (/).

```
# cd /  <cr>
#
```

13. Move "unix.nu" to the root directory.

```
# mv /usr/src/uts/unix.nu /  <cr>
#
```

14. Save the old kernel as follows:

```
# mv /unix /unix.old  <cr>
#
```

15. Make the "unix.nu" the kernel normally booted as follows:

```
# mv /unix.nu /unix  <cr>
#
```

16. Unmount the /usr file system.

```
# umount /dev/dsk/c0d0s3  <cr>
#
```

17. Update the super-block.

```
# sync;sync   <cr>
#
```

18. Reset and reboot the system.

In cases where simple changes need to be made to a terminal port's characteristics, such as altering the baud rate or its port control actions, no changes to /dev files or the generation of a new kernel would ensue.

For example, to change the baud rate of a terminal port, first determine the proper baud rate value to be used and then make the change to the desired port entry in /etc/inittab.

The following procedure shows how to change the baud rate of tty04 from 9600 baud to 2400 baud.

Note: this procedure requires the administrator to be either logged in as root or become a super-user.

PROCEDURE: How to change a terminal port's baud rate

1. Change directories to /etc.

   ```
   # cd /etc  <cr>
   #
   ```

2. Edit /etc/inittab; locate the tty04 entry and effect the change.

   ```
   04:2:respawn:/etc/getty tty04 9600155

   04:2:respawn:/etc/getty tty04 2400155
   ```

3. When done, write and quit the file.

4. Instruct the system to read and act upon the changes in /etc/inittab as follows:

   ```
   # init q  <cr>

   [ Displays various messages.]
   #
   ```

5. Perform a **ps -el** process status command to obtain the process "id" of the old "getty" on the changed port.

6. Manually kill the old "getty" on the changed port as follows:

   ```
   # kill -9 [old getty PID]  <cr>

   killed [PID]
   #
   ```

Note: changing the baud rate of the "console" terminal port requires altering the switch settings of the console port baud rate switch in the system. To obtain the proper settings, consult the ONYX 6810 MICROCOMPUTER SYSTEM USER'S GUIDE for that system.

Configuring the System for Modems

Preparing the system to use a modem device for communications between two computers or a remote terminal involves configuring both the hardware and software of the system. The hardware configuration data is explained in the ONYX 6810 MICROCOMPUTER SYSTEMS USER'S GUIDE for that particular system. Therefore, this text focuses upon the software configuration data needed to incorporate the use of a modem.

The software process may involve altering information in the /etc/inittab file, along with reprogramming a file called "dialout.c" located in the /usr/src/uucico directory.

One of the considerations for the system administrator is deciding what tty init field options are desirable when using a modem. As described earlier, the init field settings determine the state of the tty port upon logging on and off the system. When incorporating a modem, the associated tty port init field is configured to control either a "dial-in" or "dial-out" state; meaning the tty port and modem are dedicated for either accessing a remote system (dialing out) or allowing a remote system or terminal to access (dialing in) the local system.

Another consideration is whether or not the modem is "direct dial" or "auto dial." Onyx has configured the system to support both types of modems. However, the preset configuration only addresses modem products manufactured by either the "VEN-TEL" or "HAYES" companies, or modems that are directly compatible with those companies' products. To use another manufacturer's modems may involve altering the program "dialout.c," which requires a knowledge of C language programming.

The following procedure illustrates configuring tty04 to support the dial-in state at 1200 baud and the dial-out state where the baud rate is determined by the modem hardware.

PROCEDURE: **Configuring a tty port for a modem**

1. Change directories to /etc.

```
# cd /etc  <cr>
#
```

2. Edit /etc/inittab, locate the tty04 entry and effect the change as follows:

```
    [ As a dial-in state ]

04:2:respawn:/etc/getty tty04 9600155

04:2:respawn:/etc/getty tty04 1200uucp

    [ As a dial-out state ]

04:2:respawn:/etc/getty tty04 9600155

04:2:off:/etc/getty 1200
```

3. When done, write and quit the file.

4. Instruct the system to read and act upon the changes in /etc/inittab as follows:

```
# init q  <cr>

[ Displays various messages. ]
#
```

5. Manually kill the old "getty" for the changed port as follows:

```
# kill -9 [old getty PID]   <cr>

Killed [PID]
#
```

Configuring the System for cu

The cu(1C) command, as previously defined, can establish a communications link between UNIX computer systems. These "links" are generally achieved through modem devices; however, **cu** can also be used to connect between two local systems that have been "hard wired." Hard wired means that a physical connection (wire) has been made from one terminal port to another between these local systems without the use of a modem.

Note: when making a hard wired connection, the wires used for receive and transmit must be **reversed** on one end of the connection!

If **cu** is used with a modem, then the system administrator must configure the desired terminal port for modem support.

The initial setup involves configuring the "L-devices" file located in the /usr/lib/uucp directory. This file is used by both the **cu** and **uucp** facilities. The structure of the L-devices file is as follows:

EXAMPLE:

```
[type] [line] [call-device] [speed] [protocol]
```

Where:

```
       type = DIR, for direct line or ACU for automatic
              calling unit where ACU1 is a VEN-TEL
              modem and ACU2 is a HAYES modem
       line = the device name involved, such as tty03
call-device = the ACU device name or if none, contains
              a "0" as a place holder
      speed = the line speed (baud rate) of that line
   protocol = an optional field that supports other
              communication protocols
```

The L-devices file has been configured with the following preset parameters:

```
DIR ttyO1 0 1200

ACU1 ttyO1 cuaO 1200
```

Note: the actual tty "number" configured is a system dependent factor; therefore, consult the ONYX 6810 MICROCOMPUTER SYSTEM USER'S GUIDE for the system to obtain the port assignment data.

The procedures for using **cu(1C)** are described in Chapter 5, under the heading of "Establishing Communication Links."

Configuring the System for uucp

The **uucp(1C)** command provides the system administrator with the ability to establish a controlled UNIX to UNIX system communications environment.

To establish uucp communications the following files must be prepared:

- L-devices
- L.sys
- L-dialcodes

These files reside in the /usr/lib/uucp directory.

The L-devices file format has already been illustrated; therefore, no further explanation is provided.

L.sys is divided into seven fields. These seven fields include:

[name] [time] [device] [speed] [phonenumber] [login-login] [password]

Where:

name =	the system name of the remote system.
time =	the time the system may be called. If not applicable, "Any", can be placed here.
device =	the device that should be used, such as DIR or ACU, etc.
speed =	the baud rate to use.
phonenumber =	the actual telephone number used to call the remote system. May be all numbers or may have a letter prefix supplied by the L-dialcodes file.
login-login =	the login sequence accepted by the remote computer. It appears as "ogin:-ogin:" where the "1" is

intentionally left off because it is not known whether "L" or "l" has been used.

password = the password on the remote computer that is associated with this calling user. It, like the "ogin:," appears as "assword:" for the same reasons.

The L-dialcodes file is an option. Uucp functions without it as long as all telephone information is provided. However, it can make the process easier when a number of different area codes are to be used. The procedure for creating an L-dialcodes file is shown below.

PROCEDURE: **Configuring the L-dialcodes file**

1. Change directories to /usr/lib/uucp.

   # cd /usr/lib/uucp <cr>
   #

2. Create the file. Enter the desired data following the format below.

   sf 415
   ny 718
   dc 202

   [Where "sf" is the name and "415" its associated area code.]

4. When done, write and quit the file.

5. Set the file permissions as indicated below.

   # chmod 600 L-dialcodes <cr>
   #

6. Change its ownership to uucp.

   # chown uucp L-dialcodes <cr>
   #

The L.sys file initially configured on the system has been established for explanatory purposes only, and it does not illustrate the use of L-dialcodes. Therefore, an example is provided below.

A configured L.sys file might look like this:

onix1 Any ACU1 1200 sf5551212 ogin:-ogin: uucp assword: pieceofcake

Where the "sf" applies to the area code entry in L-dialcodes.

In addition, the "ogin:" fields might also be configured as:

```
ogin:--ogin:-EOT-ogin:-BREAK-ogin:
```

Where "EOT" and "BREAK" represent the <cntrl-d> and the <break> key on the terminal respectively. These keys are used for generating signals which synchronize the speed (baud rate) between the two computers.

The procedures for using uucp(1C) are described in Chapter 5, under the heading of "Establishing Communication Links."

Setting Up Synchronous Communications

In this guide, "synchronous" communications means the "Bisynchronous 2780/3780 Emulator" software package configured to operate on Onyx computer systems.

In general, this package provides the ability to interface with and transfer continuous streams of data (batched) to a mainframe class computer at a high rate of speed.

For example: A bank's branch offices might use this method to transfer their daily account activities to the home office.

It should be noted that this software is not a part of the standard release UNIX system sent with each Onyx computer. It is, however, a valuable option offered by Onyx. If acquired, it includes a tape containing the software and a detailed installation and instruction guide.

Note: a new kernel must be made after installing this software package!

Rebuilding the Operating System

This section describes how to save and make operational the configured alternate UNIX kernel.

The example procedure is brief and may be familiar since it has been explained in detail in other sections of this chapter.

After the system administrator has made the desired configuration changes, the next task to perform is generating (building) the new UNIX kernel. The following example procedure describes this process which is performed in the single-user mode.

PROCEDURE: How to rebuild the operating system

1. Mount the /usr file system.

```
# mount /dev/dsk/c0d0s3 /usr  <cr>
#
```

2. Change directories to /usr/src/uts.

```
# cd /usr/src/uts  <cr>
#
```

3. Make the new kernel as follows:

```
# make   <cr>

    touch cf/dfile
    cd cf; make "DFILE=dfile"
    /etc/config dfile
    cc -c -O -I/usr/include conf.c
    ar rv ../cflib.a conf.o
r - conf.o
    rm -f conf.o
    /lib/cpp -P -DASM  -I/usr/include m68kvec.s>m68kvec.i
    as -o m68kvec.o m68kvec.i
    ar rv ../cflib.a m68kvec.o
r - m68kvec.o
    ../cflib.a is ready

    ld -e start -o unix cflib.a iolib.a mllib.a oslib.a cflib.a
    iolib.a mllib.a oslib.a
    unix is ready
   #
```

4. Rename the new kernel.

```
# mv unix unix.nu  <cr>
#
```

5. Change directoires to root (/).

```
# cd / <cr>
#
```

6. Move the new kernel to root.

```
# mv /usr/src/uts/unix.nu  /  <cr>
#
```

7. Save the old kernel.

```
# mv /unix /unix.old  <cr>
#
```

8. Make the new kernel the one normally booted and update the super-block as follows:

```
# mv /unix.nu /unix  <cr>
#

# sync;sync  <cr>
#
```

9. Unmount the /usr file system.

```
# umount /dev/dsk/c0d0s3  <cr>
#
```

10. Update the super-block.

```
# sync;sync  <cr>
#
```

11. Press the reset button and boot up as usual.

Once the new kernel has been functionally verified, make a tape backup copy of the root file system using the **cpio(1)** command.

In the event the new kernel cannot be booted, the system administrator can boot the system from the old kernel manually. The following procedure shows how this is done.

PROCEDURE: **How to boot from an alternate kernel**

1. Press the system reset button. The console terminal should display the following:

```
-- PROM 05/03/85-12:19:48 --

0,1,c,9,?: 0  <cr>

   SHELL 02-01-85-15:45:12

SHELL$$
```

2. Perform the following:

```
SHELL$$ boot /unix.old <cr>

Booting /unix.old.

Console login:
```

Transferring System Documentation to Disk

Onyx may supply as an option with each UNIX system a tape containing the System V USER REFERENCE MANUAL package. If desired, this tape copy of the manual can be transferred into the /usr directory. However, the entire manual set consumes approximately 2 megabytes of disk storage! For systems using smaller capacity disk drives, the loss of this much storage space can significantly impair the flexibility of the system. Therefore, it is advised that only a subset of this manual be installed.

A suggested subset of this manual is: "u_man/man1," which describes most of the UNIX commands.

The following example procedure describes how to transfer this subset onto the system.

Note: this procedure requires the system administrator to be either logged in as root or to become a super-user!

PROCEDURE: **How to transfer the system manuals to disk**

1. Insert the "Manual Pages" tape into the system.

2. Change directories to /usr.

```
# cd /usr  <cr>
#
```

3. Transfer u_man/man1 from tape.

```
# cpio -ivBdu u_man/man1 < /dev/rmt_rwd  <cr>

/[directory]
 /[files]
#
```

Initiating the Multi-User Environment from Single-User

If the system is in the single-user environment, the multi-user mode can be initiated from the console terminal.

The following procedure describes how to initiate the multi-user mode.

PROCEDURE: **How to initiate multi-user mode from single-user**

1. Perform the following:

```
# init 2   <cr>

INIT: New run level: 2

Do you want to check the file system? (y or no) y <cr>

/dev/dsk/c0d0s1
file system: [name]  Volume: [name/number]
** Phase 1 - Check Blocks and sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Free List
[N] files  [N] blocks   [N] free

[ Continues to check all other file
  systems listed. ]

Current date: [date]
Process accounting started. [If enabled.]
Error logging started.
Cron started.
Line printer scheduler started. [If enabled.]

Mounted file systems -

/usr (/dev/dsk/c0d0s3): [N] blocks  [N] i-nodes
/    (/dev/dsk/c0d0s1): [N] blocks  [N] i-nodes

[ Continues to list all mounted file systems. ]

********* SYSTEM MULTI-USER [date] *********

Console login:
```

Configuring Printers

UNIX, as installed from the Onyx INIT tape, has configured both a serial and parallel printer facility.

These facilities reside in /dev and include the following:

- plp - parallel printer device file
- slp - serial printer device file
- lp - common printer driver device file

Note: as configured by Onyx, the parallel printer device file (plp) has been linked to the common printer device driver lp. Therefore, if a serial printer is to be used, the administrator must link the serial printer device file (slp) to the common printer driver lp using the ln(1) command.

When a serial printer, or any other serial device, is configured onto the system, its characteristics or "attributes" can be displayed using the **stty**(1) command. The **stty** command can also alter the attributes of a serial device. The attribute options of **stty** are quite extensive, therefore, thfe discussion on **stty** and the serial printer focusses on what the standard options are and how to make desired changes.

The following procedure shows how to view the current **stty** settings of a serial printer.

PROCEDURE: **How to view the serial printer attributes**

1. Perform the following:

```
$ stty < /dev/slp   <cr>

speed 9600 baud; -parity hupcl clocal
line = 1; -inpck -istrip -ixon -opost
-isig -icanon -echo -echoe -echok
$
```

When configuring the UNIX system to support the use of a serial printer, the administrator should set certain **stty** attributes in the /etc/rc file manually.

For example:

```
stty 300 ixany ixon ixoff onclr opost </dev/slp
```

The last aspect involving printers which may be of concern to the system administrator is enabling the desired printer for use with the scheduler facility (the spooler suggested for use). The example procedure below shows configuring the parallel (plp) printer device for use with the spooler. A serial (slp) printer device can also be easily configured by substituting **slp** for **plp** where indicated in the command syntax.

Note: this procedure requires the administrator to either log in as root or become a super-user. In addition, the printer scheduler must be turned off! This is done using the **lpshut(1M)** command.

PROCEDURE: **How to configure the parallel printer to the scheduler**

1. Change directories to /usr/lib.

```
# cd /usr/lib  <cr>
#
```

2. Add the parallel printer as follows:

```
# lpadmin -pplp -cprinter2 -mdumb -v/dev/plp  <cr>

[ Where "-p" is the printer name, "-c" is the printer
  class, "-m" is the model interface program, and
  "-v" is the actual device file name.  Note: unlike
  other command syntax, there must be no spaces
  between the option selection and the entered
  argument. ]
#
```

3. Display the new printer status as follows:

```
# lpstat -t   <cr>

scheduler is not running
system default destination: printer1
members of class printer1:
        slp
members of class printer2:
        plp
device for slp: /dev/slp
device for plp: /dev/plp
slp accepting requests since [date]
printer1 accepting requests since [date]
plp not accepting requests since [date]
      new destination
printer2 not accepting requests since [date]
      new destination
printer slp is idle.  enabled since [date]
printer plp disabled since [date]
      new printer
#
```

4. Reactivate the scheduler for use as follows:

```
# lpsched  <cr>
#
```

Note: the procedure for configuring the system to support an additional serial printer is discussed in Chapter 6, "System Expansion."

Configuring Terminals

Disparities among different manufacturers' terminals can lead to problems; UNIX circumvents the problems with "termcap," a file that resides in the /etc directory and describes certain characteristics about a wide variety of terminals. It is, as previously noted, one of the members of the Berkeley Enhancement package.

The contents of termcap typically appear as follows:

```
d0|vt100n|vt100 w/no init:is@:if@:tc=vt100:
d6|vt100|vt-100|pt100|pt-100|dec vt100:
```

The data above partially describes the features set for a "vt100" type of terminal.

If the system administrator wants to learn how to interpret the symbolism used and perhaps develop unique entries, the information is contained in the Enhanced ONYX System V PROGRAMMER REFERENCE MANUAL. However, this discussion focuses on how to use termcap.

Prior to each major terminal description section, there is a brief English definition of the terminal itself (i.e., DEC model vt100, Televideo model 925, etc.).

Using an editor's pattern matching capabilities, locate the English name of a terminal, then find its mnemonic designator such as "vt100" for a DEC model vt100. This mnemonic name is used in conjunction with a shell option to describe the terminal. The shell option is "TERM."

TERM is a parameter that can be set when building a user's .profile file. When used, TERM appears as "TERM=[mnemonic terminal name]."

Another option offered by the shell in conjunction with TERM is "EXINIT." EXINIT allows certain terminal functions to be invoked without going through the /etc/termcap file. These functions are used by the vi editor to force specific terminal screen controls to take place.

For example:

```
EXINIT='set ai '
```

Where:

ai = the auto indent feature. Forces text to be indented while entering data under **vi**.

A second terminal configuration utility is the **stty(1)** command. **Stty** is used to manipulate parameters for a serial communication device. As discussed earlier, there are many **stty** parameters that can be altered. Therefore, if a parameter is going to be changed, the system administrator should fully understand what function the parameter performs before effecting the change; otherwise, communication problems could result.

Creating the Login Environment

One of the convenient features of the UNIX system is that it permits a user to create a customized working environment which can be invoked and made a part of the normal login process.

This customization can be achieved by programming a group of special files which function either under the Bourne shell (**sh**) or the Berkeley C-shell (**csh**).

This group of files includes:

- .profile
- .cshrc
- .profile
- .login

Note: the creation or manipulation of some of these files requires a working knowledge of shell or C-shell programming.

Figure 4-1, "Basic Login Sequence," illustrates the relationships among some of these files and others involved in this operation.

Bourne Shell Special Files

The **profile** special file is constructed using shell programming conventions. When executed, **profile** implements the following actions:

1. Initiate a shell (sh).
2. Establishes basic **stty** characteristics.
3. Sets the time zone it knows.
4. Establishes the basic directory paths.
5. Displays the message of the day.
6. Inform the user of any mail.
7. Inform the user of any news.

Profile is executed every time any user logs onto the system under the Bourne shell and it can be reprogrammed to perform other functions that are deemed desirable at that time.

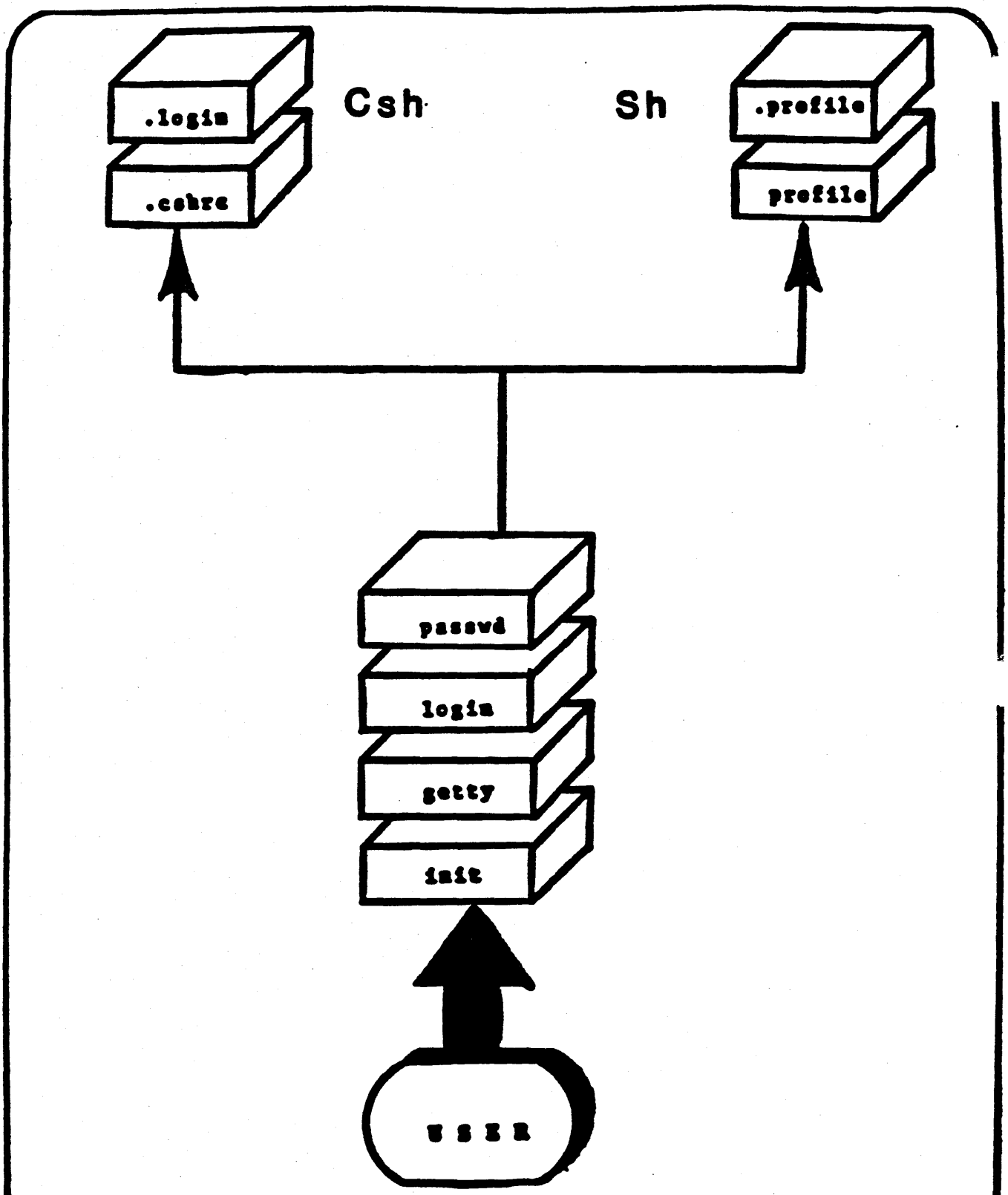


Figure 4-1 Basic Login Sequence

The **.profile** file can be programmed to establish a customized working environment for a user.

A typical **.profile** file might appear as follows:

```
EXINIT='set ai nowrapscan'
stty erase '^h' kill '^x' echoe
PATH=./:/bin:/usr/bin
SHELL:/bin/sh
PS1='[my unique prompt]'
TERM=[my terminal mnemonic in /etc/termcap]
HOME=/u/[my home directory]
MAIL=/usr/mail/[my mail stop]
export EXINIT PATH SHELL PS1 TERM HOME MAIL

[ And other special options or commands
  supported by sh. ]
```

Berkeley C-shell Special Files

The **.cshrc** file is programmed to establish a number of special conditions that are desirable prior to working on the system. Among these might be the "path" for which the system will look for commands, the user's "home" directory, or a unique "prompt" character.

For example:

```
set home=[/usr/myhome/directory]
set path=[/bin /usr/bin]
set prompt='(x)'
```

The **.login** file is read and acted upon after the **.cshrc** file. When programmed, the **.login** file performs an additional set of commands such as: "ignoreof," which instructs the shell not to exit upon receiving an end-of-file (EOF) signal from the terminal, or set the "noclobber" feature to prevent overwriting a file inadvertently.

A typical **.login** file might appear as follows:

```
set noclobber
set ignoreof
set mail=[/usr/mail/mail/me]
```

Required System Accounts

This section describes the set of required accounts that have been incorporated and configured into the system.

These accounts include the following:

- root
- uucp
- sync
- lp
- bin
- adm
- sys
- user

These accounts are "required" for one of two reasons: they are necessary for the initialization and configuration of a UNIX system, and/or they are used extensively in everyday operation.

The **root** account is essential because, like super-user, the system administrator can work in an unrestricted mode. Once entered, the **root** account allows the administrator to create file systems, verify and repair corrupted file systems, and add users, to name a few.

The other configured accounts are not as critically needed, but they are required because of their frequency of use.

Required System Directories

This section describes the directories that are required for proper operation of a UNIX system.

These directories include the following:

- bin
- dev
- lost+found
- usr
- etc
- lib
- tmp

There may be more directories residing in root, but those listed above MUST be there.

The following text describes the contents of each of these essential directories.

The **/bin** directory contains most of the executable "binary" files for the standard UNIX commands.

The /etc directory contains most of the privileged commands that can be executed or altered only by logging into root or becoming a super-user.

The /dev directory contains all the special device files for communicating with devices such as the disk drive, tape drive, and terminals.

The /lib directory contains the library of "C" language functions used by UNIX commands.

The **/lost+found** directory is used by **fsck** to store files and directories that have been orphaned. Each file system created must have this directory.

The /tmp directory is a storage place for temporary system files.

The /usr directory contains added commands, libraries, and important data for users.

Removing Unneeded Files

This section discusses the removal of tools and facilities to increase the usable disk storage space on the system.

As previously stated, the UNIX system is very flexible. One aspect of its flexibility is noted in the number of specialized facilities and tools such as "sdb," a program debugging package, which is included in the standard UNIX system. However, these "packages" may not be needed for the particular installation in everyday use. Therefore, you may decide to remove them to make more disk storage space available.

It should be noted that many of the UNIX programs call upon the resources of other programs in accomplishing their tasks. Therefore, the system administrator should determine whether or not removing the desired program impairs the function of any other program before it is removed! The Enhanced ONYX System V USER REFERENCE MANUAL is the primary source for this information. Each command description has a sub-section named "SEE ALSO;" where notations are made concerning other programs that are involved with the command in question.

To remove unneeded files, use the **rm(1)** command as described earlier in this chapter.

If a file has been removed inadvertently, it can be restored from the Onyx /usr CPIO tape using the **cpio(1)** command with the appropriate options.

Creating User Accounts

Installing a new user onto the UNIX system can be accomplished either automatically using the **adduser(1M)** command (the suggested method) or effecting the necessary changes to the appropriate system files manually.

The following discussion below pertains to adding user accounts by the automatic method using the **adduser** command. To learn how to effect the necessary changes manually, refer to Section 5, "Creating Additional User Accounts" in this guide.

Note: using the **adduser** command requires the administrator to be either logged in as "root" or become a super-user.

PROCEDURE: **How to use the adduser program**

1. To add a user, perform the following:

   # adduser <cr>

   This program is used to create a new user account on the computer. You are prompted for all of the particulars and after answering all prompt, a new user account and login directory is created. Enter <Del> if you choose to exit at any time...

   Each user login requires a unique name. This name must be 8 character or less and should be one that is easily remembered...

   Enter the new user name: [**myname**] <cr>
   User = myname
   Ok? (y/n): **y** <cr>

   Scanning for next available user id number.

   The next available user id is [number]
   Press <Return> to use or enter a new id: <cr>

   Scanning for the default Group id number.
   The default Group id is [number]
   Press <Return> to use or enter a new id: <cr>

   Group #: [number]
   Group Name: [name]
   Ok? (y/n): **y** <cr>

```
User accounts also contain a description of
what a particular user login id is for.

Enter a description for user myname: workspace  <cr>
Description = workspace
Ok? (y/n): y  <cr>

Make the user's HOME login directory.
Example: /u/myname for
user myname

Enter a login directory for user myname: myname  <cr>
Directory = myname
Ok? (y/n): y  <cr>

Set the user's login shell
Selection:
            Standard Unix Shell - /bin/sh
            Berkeley "C" Shell  - /bin/csh
         ONYX Office Main Menu  - mainmenu

Enter a login shell for user myname: /bin/sh <cr>
Shell = /bin/sh
Ok? (y/n): y  <cr>

The following has been entered:

1. User Name    = myname
2. User ID #    = [number]
3. Group ID #   = [number]
4. Description = workspace
5. Directory    = myname
6. Shell        = /bin/sh

Do you want to continue and add myname? (y/n): y <cr>

[ If answered "y," the program effects the necessary
  changes to the system; if answered "n," it explains
  how to make changes to any entry made above. ]

Directory myname created for user myname ...

Assign a password for the user's account.
The password should be at least 6 characters long.
Assign a password for myname? (y/n): y  <cr>

[ If answered "n," the program displays..]

Ensure user myname, adds a password when convenient!!!

[ Otherwise, the program displays...]
```

```
Changing password for myname
enter password: [password selected] <cr>
re-enter the password: [enter it again] <cr>

Add another user? (y/n): n  <cr>

adduser program exited...
#
```

If a user forgets his/her password, the system administrator can, using super-user mode, delete the password field in /etc/passwd. The user can then make a new password manually using the **passwd**(1) command.

Setting Up Automatic System Functions

There are numerous functions that occur automatically during normal everyday use of the system.

However, there are three specific functions a system administrator should become familiar with:

- Automatic startup -- /etc/rc
- Automatic shutdown -- /etc/shutdown
- Automated command execution -- /usr/lib/crontab

In review, rc is a programmable file whereupon initiating multi-user mode will start up the programs necessary for this environment.

The **shutdown** program performs an organized and safe transition from the multi-user to the single-user environment. The system administrator executes **shutdown** by entering "/etc/shutdown" from the console terminal while logged in as root or as a super-user. The actual procedure is shown in chapter 5, "The Daily Routine."

Lastly, **crontab** is a file that is read and executed every minute by the /etc/cron program which is initiated by /etc/rc. The **crontab** file allows the administrator to program a set of events to occur at a specified time. For example, to instruct the system to clean out the contents of the /usr/adm/sulog file on Sunday October 6, at 11:55pm, construct the **crontab** file as follows:

```
55  22  6  10  0  cat /dev/null>/usr/adm/sulog%
```

[Where:

```
55 = the minutes 0-59
22 = the hours 0-23
 6 = the numeric day of the month 1-31
10 = the numeric month of the year 1-12
 0 = the numeric day of the week 0-6, "0" is Sunday ]
```

In addition, **crontab** supports a set of special functions which add flexibility to its use. This set includes: the **dash** (-), the **asterisk** (*), the **comma** (,), and the **percent sign** (%).

The **dash** (-) allows a "range" of numeric values to be established and acted upon for any numeric field in **crontab** (i.e., 55-59 minutes.).

The **asterisk** (*) represents all the legal numeric values supported in **crontab** (i.e., 0 to 59 minutes.).

The **comma** (,) is used to separate two consecutive numeric values within a **crontab** field (i.e., 55,59 minutes.).

The **percent** sign (%) is used to simulate a **<cr>** function in a **crontab** command string.

Note: the **crontab** file can also be used to automate tasks such as system backup and the removal of unwanted files, to name a few.

Adding other Software Packages

This section describes adding "special" (not part of UNIX) software packages.

Special software packages include the following:

- Database management systems
- Word processors
- Financial application tools
- Graphics display tools
- Languages such as FORTRAN, COBOL, BASIC

Many such packages enhance the system's flexibility, productivity, and efficiency. However, with the prospective addition of any of these packages, there are a number of considerations that must be taken into account.

First, the system administrator should determine whether or not there is enough disk storage space to accommodate this package. Most of these packages indicate how much disk space they take up. The **df(1)** command shows how much space is left on the file system (usually the "root") where this package will reside.

Second, most of these packages come with a detailed instruction and installation document. These documents should be read thoroughly before attempting to integrate the software into the system!

Third, some of these packages may involve altering the basic system configuration. Therefore, it is prudent to make a copy of the root file system as it is currently configured before installing the additional software. A copy of the root file system can be generated using the **cpio(1)** command. It is also a good idea to make a current backup of the user's file system(s) if one does not already exist.

Finally, after the software package has been added, the system administrator should monitor the system's response and performance for a given period of time to determine whether or not any adverse effects have resulted.

Protecting the System

This section describes a set of rules the system administrator can follow to maintain a reasonable level of system security.

First, and most importantly, the system administrator should assign a password to the root login account and all other required accounts!

Second, the system administrator should be careful when altering the permissions of any of the files in the root file system. Remember, "permissions" means the method used to control a user's access to a file or directory and what a user can do with the file.

Third, create "casual" user accounts only when necessary and ensure that their access in terms of where they can go and what they can do is restricted. Again, this is done with proper file permissions, controlling the **.profile** settings, and possibly the **rsh(1)** restricted shell feature of the Bourne shell.

The restricted shell feature is implemented by directing the user's path through /bin/rsh instead of /bin/sh as established in the /etc/passwd file.

For example:

```
casual::10:10:Casual User:/u/casual:/bin/rsh
                                      :
                                [ was /bin/sh ]
```

Next, the system administrator should not allow users to perform any work in the root file system. As previously discussed, users should do their work in other file systems.

Finally, the system administrator should ensure that all users make and keep secret their login passwords. Passwords are effective if they are known **only** by their intended users!

Setting Up System Accounting

The system accounting facility in UNIX can provide a massive amount of statistical data relating to command usage by users, how long a user was logged onto the system, and disk usage resource information.

The elements of the accounting facility can be divided into three general groups: those which generate reports, those which initiate main actions, and those which provide special functions.

Under report generation, there are eight programs which provide various types of reports in various formats. These programs all reside in the /usr/lib/acct directory.

They are:

- acctcms - provides command usage data.
- acctcon1/2 - provides connect time data.
- acctprc1/2 - provides process accounting data.
- monacct - provides periodic file cycling and monthly summary data.
- prctmp - provides a printout of session related data.
- prtacct - provides a printout of consolidated accounting data.
- prdaily - provides a report of the previous day's accounting data.
- runacct - includes the main daily accounting functions.

The following programs initiate main actions in the accounting facility. These commands reside in the /usr/lib/acct directory.

They are:

- turnacct (shutacct) - turn accounting on/off.
- accton - activate line connect accounting.
- startup - command process accounting.
- lastlogin - update user login trace file.
- acctwtmp - update login/logoff history file.

The following programs provide special information or control in the accounting facility. These programs reside in the same directory as the foregoing.

They are:

- acctcom - search for and print accounting data.
- acctdusg - compute disk resources.
- acctdisk - create disk usage totals for consolidation.
- acctmerg - merge summary accounting data.
- chargefee - enter a special account billing charge.
- ckpacct - monitor size of accounting files.
- dodisk - perform time initiated disk accounting function.
- fwtmp - create ASCII accounting records for editing.
- wtmpfix - adjust accounting records for clock changes.

The data gathered by accounting is kept in sub-directories named "fiscal," "nite," and "sum" which reside in the main directory /usr/adm/acct. In addition, two more accounting files "pacct" and "wtmp," reside in the /usr/adm directory.

The "pacct" file contains process termination data that is generated by the kernel, and "wtmp" keeps a record of the login and init activities.

For convenience, the accounting facility can be configured and initialized upon entering the multi-user environment by programming the /etc/rc file.

Making a Copy of the System

This section describes how to make a backup copy of the root file system. As expressed earlier, a copy of the root file system should be made after the system has been configured and whenever it is deemed desirable by the system administrator.

The following procedure shows how to make a copy of the root file system. The system should be in the single-user environment and the file system's integrity should be checked using the **fsck(1M)** command. If the check showed errors, fix them prior to making the backup copy.

Note: this procedure must be performed in the single-user environment!

PROCEDURE: How make a copy of the root file system

1. Using a cartridge tape, ensure it is NOT in the SAFE position, then insert the tape into the system.

2. Change directories to root (/).

```
# cd /  <cr>
#
```

3. Perform the following:

```
# find . -print |cpio -ovcB > /dev/rmt_rwd  <cr>

[ Where the find command string instructs the system
  read all the files in all the root directories recursively.
  The "ovcB" command options mean the following: o=output
  mode, v=verbose display mode, c=writes a header in ASCII,
  and B=blocked output at 5120 bytes-per-block.  The
  "rmt_rwd" means the cartridge tape device -- write all
  tracks. ]

/[directories]
 /[files]

[Total number of blocks for files]

#

[ The copy may take 20 minutes or more. ]
```

It is important to label these tapes with the name of the file system, when the backup was performed, and the method used (cpio with its options) to create the backup. Ensure the tape is set to the SAFE position and store it in a dry cool place to preserve data integrity.

Note: the **cpio(1)** utility has many options that can be envoked. Because of its flexibility, **cpio** is focused upon extensively in this guide for transferring files and directories between disk and tape. Therefore, it is suggested that the system administrator become very familiar with this UNIX utility!

Documenting the System

The system administrator needs to keep a physical record of the system's layout and parameters. This record or "log" should include, but not be limited to, the following:

1. Physical Layout

   (a) Number of terminal ports
   (b) What users are on what terminal port
   (c) Ports dedicated to modem communications
   (d) Ports dedicated to parallel printers
   (e) Ports dedicated to serial printers
   (f) Number and types of disk drives
   (g) Number of tape drives

2. Logical Layout

   (a) File systems

      (1) How many
      (2) Their names
      (3) Their size
      (4) Their general purpose
      (5) Their location on the disk

   (b) The number and names of user accounts on the system
   (c) Number and names of any restricted user accounts
   (d) The names of and places where any additional software packages have been incorporated
   (e) The name and function of any installed special device driver

3. Maintenance Log

   (a) System model number
   (b) System serial number
   (c) Maintenance contract number (if any)
   (d) Who to call for service
   (e) Record of maintenance performed
   (f) System down time

4. Data Backup Log

   (a) Date performed
   (b) What file system was backed up

5. System Trouble Log

   (a) When it occurred
   (b) Who reported the problem
   (c) Symptoms
   (d) Correction
   (e) When corrected
   (f) Is it a recurring problem?

6. System Accounting Statistics (if applicable)

7. Security Breach

   (a) Who reported it
   (b) Who committed it
   (c) How was it committed

8. General Information

   (a) Mail sent to users
   (b) News sent to users
   (c) Planned system events

The outline seems large and time consuming; however, it can actually save time in the event that a problem arises.

Summary

This chapter educates the system administrator on the following:

- Generating the operating system
- Transferring software between tape and disk
- Checking and maintaining system integrity
- Configuring the system
- System communications techniques

This chapter also describes what is required for system operation, and it outlines a list of general practices the system administrator can follow.

In addition, this chapter illustrates the following procedures:

- How to transfer the root file system
- How to auto-configure the devices and /usr file system
- How to boot up the system for normal use
- How to use the fsck(1M) command
- How to use the sysdef program
- How to use the diskconf program
- How to use the devices program
- How to make the file system

- How to use the labelit command
- How to install a new device driver
- How to configure and install additional tty ports
- How to change a terminal port's baud rate
- Configuring a tty port for a modem
- Configuring the L-dialcodes file
- How to rebuild the operating system
- How to boot from an alternate kernel
- How to transfer the system manuals to disk
- How to initiate multi-user mode from single-user
- How to view the serial printer attributes
- How to configure the parallel printer to the scheduler
- How to use the adduser program
- How make a copy of the root file system

TABLE OF CONTENTS

CHAPTER 5

CHAPTER 5

THE DAILY ROUTINE

Preface

This chapter describes and illustrates those tasks that are routinely performed by the system administrator on a daily basis.

The text associated with each topic has been kept brief to draw more attention towards the actual procedure.

Remember, while following a procedure all commands and actions requested appear in **bold type.** Any system response to a requested command appears immediately below that command. In addition, all comments made within a procedure are contained between brackets "[]." The symbol **<cr>** means "carriage return."

Carefully read all **Cautions** and **Warnings.** They provide important information.

Note: the system responses appearing herein are general and may differ among systems.

Startup

The following outlines the system initialization procedure which may occur each day. The example presumes a "cold start" condition; that is, the machine was turned off and needs to be powered on before initialization. If it has not been turned off, then the bootup portion should not be performed.

PROCEDURE: **How to initialize the system**

1. Apply power to the system. The console terminal should display the following:

   -- PROM 05/03/85-12:19:48 --

   0,1,c,9,?: 0 **<cr>**

   SHELL 02-01-85-15:45:12
   SHELL$$

```
[ Note: approximately one minute elapses between the
  "PROM" and "SHELL" messages.  This time is used for
  running the self test programs.  Unless an
  error occurs, no messages are displayed during
  the self test phase. ]
```

2. Perform the following:

```
SHELL$$ boot  <cr>
Booting /unix.

Console login: [your login name]  <cr>

password: [your password]  <cr>

Do you want to check the file system? (y or no) y <cr>

/dev/dsk/c0d0s1
file system: [name]  Volume: [name/number]
** Phase 1 - Check Blocks and sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Free List
[N] files  [N] blocks   [N] free

[ Continues to check all other file
  systems listed. ]

Current date: [date]
Process accounting started.  [ If enabled ]
Error logging started.
Cron started.
Line printer scheduler started.  [ If enabled ]

Mounted file systems -

/usr (/dev/dsk/c0d0s3): [N] blocks  [N] i-nodes
/    (/dev/dsk/c0d0s1): [N] blocks  [N] i-nodes

[ Continues to list all mounted file systems. ]

********** SYSTEM MULTI-USER [date] **********

#
```

Shutdown

This procedure illustrates the way in which a system administrator should terminate multi-user mode after work has ceased, or at any other time when it is necessary to return to single-user mode.

PROCEDURE: **How to use the shutdown program**

1. From the console terminal, log in as root, and execute the shutdown program.

   ```
   # /etc/shutdown   <cr>
   #
   ```

2. The following is displayed on the console terminal. Answer the self-explanatory questions.

   ```
   SHUTDOWN PROGRAM
   Do you want to send your own message? (y or n):y  <cr>
   Type your message followed by ctrl d....

   [ The program notifies ALL users that the system
     is coming down in 60 seconds. ]

   SYSTEM BEING BROUGHT DOWN NOW ! ! !

   Busy out (push down) the appropriate phone lines for this system.
   Do you want to continue? (y or n):y  <cr>

   Line printer scheduler stopped.  [ If enabled ]
   Process accounting stopped.  [ If enabled ]
   Error logging stopped.

   All currently running processes will now be killed.

   ****   SYSCON LINKED TO /dev/console   ****

   ****   INIT s EFFECTIVE IN 20 SECONDS  ****

   Wait for 'INIT: SINGLE USER MODE' before halting.

   INIT: New run level: S

   INIT: SINGLE USER MODE

   Current date:[date]

   #
   [ The error messages that shutdown may
     exhibit are:
     For help, call your system administrator.
     Only the shutdown messsage has been sent.
     THE SYSTEM IS STILL IN MULTI-USER STATE! ]
   ```

3. The system is now in single-user mode.

System Disk Management

Monitoring the disk storage means to: find out how much disk storage is used, find out how much disk storage is available, purge data files, and align the file system.

How Much is Used/Available

The following procedure shows how to determine how much disk storage space is used/available. While performing these procedures, the system administrator should be either logged in as root or become a super-user.

PROCEDURE: **How to determine disk storage usage**

1. Check disk storage as follows:

```
# df  <cr>

/  (/dev/dsk/c0d0s1): 2523 blocks   4293 i-nodes

[ Lists all file systems contained in
  /etc/mountable ]
#
```

If the "blocks" value displayed is 100 or less, the system administrator must free up more space on that file system! The system administrator can determine further the directory that is too large by using the **du(1)** command.

PROCEDURE: **How to further isolate disk storage usage**

1. Find the directory that is using too much storage as follows:

```
# du [/file system] [/directory] <cr>

[blocks] /[subdirectory]
[blocks] [files]
[total blocks used] /[main directory]
#
```

Once the directory in question is identified, the system administrator should notify and instruct the directory's owner to remove all unnecessary files.

Data File Purging

If lack of disk storage space is in the root directory, first check the /tmp directory and remove all the files.

Note: this procedure should be performed in the single-user mode!

PROCEDURE: **How to purge data files**

1. Change directories to /tmp, list the contents, then remove the files as follows:

```
# cd /tmp  <cr>
#

# ls -l  <cr>

rwxrwxrwx 1 root system [size][date/time] file name
#

# rm -f [file name(s)]  <cr>
#
```

Next, check the contents of the following files in the /usr/adm directory: "pacct," "sulog," and "wtmp." These files are used by the system to accumulate statistical data on system activity. Therefore, over time they can grow quite large. If the contents are of value print them onto a printer; otherwise remove the contents using the procedure below.

Note: the "pacct" file is a data-type file which does not allow printing onto a printer.

PROCEDURE: **How to use the null device to purge files**

1. Change directories to /usr/adm. Display the contents of each file as follows:

```
# cd /usr/adm   <cr>
#

# more -d [file name] <cr>

[ Displays contents of a file a page
  at a time. ]
#
```

2. Purge the file contents as follows:

```
# cat /dev/null > [file name]  <cr>
#

[ This removes the contents, but leaves
  all other aspects of the file alone. ]
```

File System Realignment

Files are written onto the disk contiguously, if possible, and their location and the number of blocks left for allocation is maintained in the system's "free list." However, over time the free list organization on the disk becomes scattered (non-contiguous). Once this happens, it takes a greater amount of time to locate and reassemble a file. By reconstructing a file system's "free list" the system regains the contiguity.

Note: to regain the contiguity of file organization on the disk requires backing up all the files in a file system onto tape, and then restoring them back onto disk.

Note: while performing this procedure, the system should be in single-user mode!

PROCEDURE: **How to rebuild the system free list**

1. Rebuild the free list as follows:

```
# fsck -S1   <cr>

/dev/dsk/c0d0s1
file system:[name]  Volume:[name/number]
** Phase 1 - Check Blocks and sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Count
** Phase 5 - Check Free List (Ignored)
** Phase 6 - Salvage Free List
[N] files [N] blocks [N] free

***** BOOT UNIX (NO SYNC!) *****
```

[The "-S" option instructs the program to reconstruct the free list only upon detecting no errors during the check. If errors were detected, the program terminates immediately. The "1" denotes the desired file system.]

2. Type nothing further on the terminal! Press the reset button, and boot up the system.

Backing Up and Restoring the System

This section describes how to save and restore an entire file system, a directory, or a single file.

Saving and Restoring an Entire File System

This procedure uses the **cpio(1)** command to save and restore data between disk and tape drives.

Note: check the file system for errors then mount it before saving the data onto disk!

PROCEDURE: **How to back up a file system**

1. Using a cartridge tape, ensure it is not in the SAFE position, then insert it into the system.

2. If the file system is not the root, then mount it as described below; otherwise, skip this step.

   # mount /dev/dsk[number] [/name] <cr>
   #

3. Change directories to the file system being saved.

   # cd /[file system name] <cr>
   #

4. Save the file system as follows:

   # find . -print |cpio -ovcB > /dev/rmt_rwd <cr>

   [Where the **find** command string instructs the system to read all the files in all the root directories recursively. The "ovcB" command options mean the following: o=output mode, v=verbose display mode, c=writes a header in ASCII, and B=blocked output at 5120 bytes-per-block. The "rmt_rwd" means the cartridge tape device -- write all tracks.]

   /[directories]
   /[files]

   [Total number of blocks for files]

   #

   [The copy may take 20 minutes or more.]

It is important to label these tapes with the name of the file system, when the backup was performed, and the method used (cpio and the options used) to create the backup. Ensure the tape is set to the SAFE position and store it in a dry cool place to preserve data integrity.

To restore a file system, follow the procedure described below in the single-user environment.

PROCEDURE: **How to restore a file system**

1. If the file system to be restored is not the root, mount that file system as previously described.

2. Ensuring the tape to be restored is in the SAFE position, insert it into the system.

3. Restore the file system as follows:

```
# cpio -ivcBdu < /dev/rmt_rwd  <cr>

[ Where the options envoked mean: i=input mode,
  v=verbose display mode, B=blocked at 5120
  bytes-per-block, d=create directories as needed,
  and u=copy unconditionally. ]

/[directories]
 /[files]

[Total number of blocks transferred]
#
```

4. Update the super-block.

```
# sync;sync  <cr>
#
```

5. If the file system was mounted, and not the root, unmount it as follows:

```
# umount /dev/dsk/[file system device name] <cr>
#
```

Saving and Restoring a Directory or File

The contents of a directory or file are saved or restored using the **cpio(1)** command as described in the following procedures.

PROCEDURE: **How to save a directory using cpio**

1. Change directories to the one being saved.

   # cd [/directory name] <cr>
   #

2. To save the directory, perform the following.

   # find . -print | cpio -ovcB > /dev/rmt_rwd <cr>

   [Where the find command string instructs the system to read all files in that directory (sub-directory) recursively. The "ovcB" command options mean the following: o=output mode, v=verbose display mode, c=writes a header in ASCII, and B=blocked output at 5120 bytes-per-block.]

   [Displays the list of file(s) and blocks being transferred to tape.]
   #

PROCEDURE: **How to restore a directory using cpio**

1. Change directories to the one being restored.

   # cd [/directory name] <cr>
   #

2. To restore the data, perform the following.

   # cpio -idcumvB < /dev/rmt_rwd <cr>

   [Where the options "idcumB" mean: i=input mode, d=create directories if needed, c=writes a header in ASCII, u=copy unconditionally, m=retain previous file modification time(s), B=blocked input at 5120 bytes-per-block, and v=verbose display mode.]

   [Displays list of file(s) and blocks being transferred to disk.]
   #

PROCEDURE: **How to save a single file using cpio**

1. To save a file, perform the following:

   # echo [/directory/file] | cpio -ovcB >/dev/rmt_rwd <cr>

   [file name]
   [blocks used]
   #

Note: the procedure for restoring a single file is the same as restoring a directory using **cpio** except the desired file name is added to the command string just after the **cpio** options (i.e., cpio -idcumvB file name </dev/rmt_rwd).

Saving and Restoring Data on Individual Tape Tracks

The procedure illustrate above, used the **rmt_rwd** cartridge tape device file. This special device file addresses and utilizes the tape as one continuous track. The special device files for addressing and utilizing the individual tape tracks are contained in the /dev/rcmt directory and appear in the following format:

c1d0t0 or c1d0t0n

Where:
c1 = the controller bus address.
d0 = the device number.
t0 = the actual track number.
In this case, "0."
n = the flag for a no rewind.

The following procedure shows how to write and retrieve a file to and from track "0" of the cartridge tape using the **cpio(1)** utility.

PROCEDURE: **How to write and retrieve a file on a tape track**

1. To write a file to track 0, perform the following:

 # echo[file name]|cpio -ovcB >/dev/rcmt/c1d0t0 <cr>

[file]
[blocks used]
#

2. To retrieve a file, perform the following:

 # cpio -ivBdu[file name] < /dev/rcmt/c1d0t0 <cr>

[file]
[blocks used]
#

Adding or Removing a File or Directory

This section describes how files or directories are created and removed.

It should be noted that users can freely create or remove files and sub-directories which reside under and are owned by the user's principal (home) directory. However, for a system

administrator to remove such files and sub-directories, the administrator needs to either log in as root or become a super-user.

Directories

The following procedure shows how to use the **mkdir(1)** (make a directory) and **rmdir(1)** (remove a directory) commands.

Note: the files residing under a directory or sub-directory must first be removed before removing the actual directory or sub-directory. This is accomplished using the **rm(1)** command with the "-ir" options (e.g., rm -ir <cr>).

PROCEDURE: **How to create and remove a directory**

1. Change to the desired file system.

   # cd [/file system] <cr>
   #

2. To create a directory, perform the following:

   # mkdir [directory name] <cr>
   #

   [OR]

   [To remove a directory, perform the following:]

   # rmdir [directory name] <cr>
   #

Files

This section illustrates the **rm(1)** command and describes the procedures for copying and moving a file using the **cp(1)** and **mv(1)** commands, respectively. In addition, this section shows the method for creating a "null" file meaning a file that has a file name but no contents.

PROCEDURE: **How to create a null file**

1. To create a null file, do the following:

   # cat /dev/null >[file name] <cr>
   #

PROCEDURE: **How to copy a file**

1. To make a copy of a file, do the following:

```
# cp [file 1] [to file 2]  <cr>
#
```

[Note: file 2 must have a different name from file 1.]

PROCEDURE: **How to move a file**

1. The **mv(1)** command can be used to overwrite an existing file, rename/create a new file, or move a file to another directory. An example of each procedure is shown below.

[Move a file to another directory]

```
# mv [myfile] [/directory]  <cr>
#
```

[Rename/create a new file]

```
# mv [myfile] [newfile]  <cr>
#
```

[Overwrite an existing file]

```
# mv [file 1] [file 2]  <cr>
#
```

PROCEDURE: **How to remove a file**

1. To remove a file, perform the following:

```
# rm -f [file name]  <cr>
#
```

[Note: if the file is not in the current working directory, use full pathnames.]

Changing Permissions

This section describes the use of: **chmod(1)** (change mode,) **chown(1)** (change ownership,) and **chgrp(1)** (change group) commands.

Again, users can perform these functions on what is owned by them; however, if the system administrator is going to change the permissions of a user's file, the administrator must either log in as root or become a super-user.

The definitions of the fields involved are:

```
-rwxrwxrwx [n] [owners name] [group name][file name]
  :  :  :
 [1][2][3]
```

Where:

r = means READ permission with an octal of 4
w = means WRITE permission with an octal value of 2
x = means EXECUTE permission with an octal value of 1
1 = OWNERS or user's permissions field
2 = GROUP permissions field
3 = WORLD or others permissions field

The **chmod(1)** command uses the octal representations for read, write, and execute when changing a file's permissions. In the command syntax, these octal values will be "additive" for each individual field.

For example, changing the owner/user field to reflect permissions of read, write, and execute would be expressed using an octal value of 7; whereby "7" was derived by adding the octal values of 4 (read), 2 (write), and 1 (execute) together.

The following example procedure shows how to use these commands.

PROCEDURE: **How to change a file's permissions**

1. Change permissions of a file to read, write, and execute for all fields.

   # chmod 777 [file or directory] <cr>
   #

2. Alter the group and others fields read permissions only.

   # chmod 744 [file or directory] <cr>
   #

   [it would look like: rwxr--r--]

Note: all directories should have execute (x) permissions; otherwise, their contents cannot be accessed!

PROCEDURE: **How to change a file's ownership**

1. Change the user ownership of a file or directory as follows:

```
# chown [new owner] [file or directory] <cr>
#
```

PROCEDURE: **How to change the group ownership of a file**

1. Change the group ownership of a file or directory as follows:

```
# chgrp [new group] [file or directory] <cr>
#
```

Killing Processes

This section describes how to terminate or "kill" a system process manually.

To determine what processes are running on the system at any given time, the system administrator uses the **ps(1)** process status command. This command has many options which control the format presentation of its information. For this purpose, the option "-el" is used because it provides a detailed display such as that shown in the following example:

```
F S UID  PID PPID  C PRI NI ADDR SZ  WCHAN TTY TIME CMD
3 R 7    234 47   67 83  20 241  2   73610 co  0:01 getty
```

Where:

F = Flags field indicators: 1=in core memory; 2=system process; 3=in core and system process; 4=locked in core; 10=being swapped-out
S = State of the process: O=nonexistent; S=sleeping; W=waiting; R=running; I=intermediate; Z=terminated; T=stopped; X=growing
UID = User identification number
PID = Process identification number
PPID = Parent/child process identification number
C = Process utilization scheduling
PRI = Priority of the process. Higher [N] means lower priority
NI = Nice value, user invoked with the **nice(1)** command
ADDR = Memory address
SZ = Number of blocks in memory the process consumes
WCHAN = The event for which the process is waiting
TTY = The controlling terminal
TIME = The cumulative execution time
CMD = The command being executed

For killing processes, the fields of concern in the display are the process identification number (PID) and parent/child process identification number (PPID).

```
************************ CAUTION ***********************************
*                                                                  *
* If the process being killed has children, the children must      *
* be killed before the parent can be killed.  Otherwise, a         *
* child process may become orphaned which may create other         *
* problems.                                                        *
*                                                                  *
********************************************************************
```

PROCEDURE: **How to kill a process**

1. Kill a process as follows:

   ```
   # kill -9 [PID or PPID number]  <cr>
   killed: [PID or PPID number]
   #
   ```

   [Note: using **kill** option "0" instead of "9" terminates all child processes associated with a named parent automatically.]

If the system administrator needs to kill all system processes quickly, this can be done using the **killall** program. To use it, the system administrator must be either logged in as root or become a super-user from the console terminal.

PROCEDURE: **How to kill all processes simultaneously**

1. Kill all processes quickly by doing the following:

   ```
   # /etc/killall   <cr>
   Killed: [PID numbers]
   #
   ```

Note: if a process cannot be terminated as described above, it may then be necessary to reboot the system.

Maintaining System Documentation

This section describes how to maintain the system administrator "log," previously mentioned in Chapter 4.

The system administrator should maintain a current record including, but not limited to, the following:

1. Changes in the physical layout of the system

   (a) Additional peripheral devices
   (b) Deleted peripheral devices
   (c) Moved peripheral devices

2. Changes in the logical layout of the system

   (a) Added file systems
   (b) Altered file systems
   (c) Added user accounts
   (d) Deleted user accounts
   (e) Added software packages, or features

3. Data backup and restore activity

   (a) When performed
   (b) What was saved or restored
   (c) Next date to perform

4. System accounting

   (a) What was gathered
   (b) What was saved or deleted
   (c) What abnormalities, if any, were indicated

5. System problems

   (a) What type of problem
   (b) Who reported it
   (c) How resolved
   (d) When resolved
   (e) If not resolved, why?

6. Maintenance

   (a) When performed
   (b) Type performed
   (c) What was performed
   (d) Total down time
   (e) When returned to operation

7. General information

   (a) Mail sent to users
   (b) News sent to users
   (c) Synopsis of content
   (d) Date sent

Creating Additional User Accounts

This section shows two methods for adding a user to the system. These two methods are:

- Adding users automatically
- Adding users manually

Adding Users Automatically

The **adduser(1M)** command allows the administrator to perform the tasks necessary for adding a user to the UNIX system such as: creating a login name, setting the account's user-id (uid) and group-id (gid), and establishing the account's home directory, login shell and password, automatically! The **adduser** program prompts the administrator for each parameter and explains briefly the use of these parameters. The **adduser** program resides in the /etc directory.

Note: making changes such as these to the system requires the administrator to either login as "root" or become a super-user.

PROCEDURE: **How to use the adduser command**

1. To add a user automatically, perform the following:

   # adduser <cr>

   This program is used to create a new user account on the computer. You are prompted for all of the particulars and after answering all prompt, a new user account and login directory is created. Enter <Del> if you choose to exit at any time...

   Each user login requires a unique name. This name must be 8 character or less and should be one that is easily remembered...

   Enter the new user name: [**myname**] **<cr>**
   User = myname
   Ok? (y/n): **y <cr>**

   Scanning for next available user id number.

   The next available user id is [number]
   Press <Return> to use or enter a new id: **<cr>**

   Scanning for the default Group id number.
   The default Group id is [number]
   Press <Return> to use or enter a new id: **<cr>**

```
Group #: [number]
Group Name: [name]
Ok? (y/n): y  <cr>

User accounts also contain a description of
what a particular user login id is for.

Enter a description for user myname: workspace  <cr>
Description = workspace
Ok? (y/n): y  <cr>

Make the user's HOME login directory.
Example: /u/myname for
user myname

Enter a login directory for user myname: myname  <cr>
Directory = myname
Ok? (y/n): y  <cr>

Set the user's login shell
Selection:
            Standard Unix Shell - /bin/sh
            Berkeley "C" Shell  - /bin/csh
         ONYX Office Main Menu  - mainmenu

Enter a login shell for user myname: /bin/sh <cr>
Shell = /bin/sh
Ok? (y/n): y  <cr>

The following has been entered:

1. User Name    = myname
2. User ID #    = [number]
3. Group ID #   = [number]
4. Description = workspace
5. Directory    = myname
6. Shell        = /bin/sh

Do you want to continue and add myname? (y/n): y <cr>

[ If answered "y," the program effects the necessary
  changes to the system; if answered "n," it explains
  how to make changes to any entry made above. ]

Directory myname created for user myname ...

Assign a password for the user's account.
The password should be at least 6 characters long.
Assign a password for myname? (y/n): y  <cr>

[ If answered "n," the program displays..]

Ensure user myname, adds a password when convenient!!!
```

```
[ Otherwise, the program displays...]

Changing password for myname
enter password: [password selected] <cr>
re-enter the password: [enter it again] <cr>

Add another user? (y/n): n  <cr>

adduser program exited...
#
```

Adding Users Manually

In the event a user account cannot be added by using the **adduser** command, the following procedure describes how to add users manually.

PROCEDURE: **How to add user accounts manually**

1. Change directories to /etc and edit the "passwd" file to create a new entry.

   ```
   # cd /etc   <cr>
   #

   # [edit] passwd  <cr>

   [1]::[2]:[3]:[4]:/[5]/[6]:[7]

   [ Where:
       1 = log in name
       2 = user identification number
       3 = group identification number
       4 = user name
       5 = file system
       6 = user's directory
       7 = shell or program ]
   ```

2. When done, write and quit the file. If a user is going to belong to more than one "group," an entry must be made in the /etc/group file.

3. Change directories to the file system where the user directory is to be made, and make the directory.

   ```
   # cd [file system]  <cr>
   #

   # mkdir [user directory name]  <cr>
   #
   ```

4. Change directories to the user directory, and create a basic .profile file.

```
# cd [user directory]  <cr>
#

# [edit] .profile  <cr>

stty erase '^h' kill '^x' echoe
TERM=[terminal type]
PATH=/bin:/usr/bin
HOME=[/filesystem][user dirname]
MAIL=/usr/mail/[login name]
export TERM PATH HOME MAIL
```

5. When done, write and quit the file.

6. Have the new user log in and set his/her password.

```
<cntrl-d>

login:[login name]  <cr>

$ [ or % ]

$ passwd   <cr>

Changing password for [login name]
enter password: [password selected] <cr>
re-enter the password: [enter it again] <cr>
$
```

Creating Turnkey Accounts

The following section describes "turnkey" accounts and how they differ from a normal user account. It also illustrates how to create a turnkey account.

A turnkey account immediately places the user in an application environment upon logging in. The application environment can be a database management system (DBMS), a word processing system, or a financial management system, to name a few. A turnkey account differs from a normal user account in that the UNIX environment is masked or appears transparent in operation to the turnkey account.

A turnkey account is created in the same manner as a normal user account. However, the pathname information needed to direct this account into the special application is substituted for the shell path direction as shown in the procedure below:

PROCEDURE: How to create a turnkey account

1. Create the account as normal by editing the /etc/passwd file.

   ```
   # [edit] /etc/passwd  <cr>
   #
   ```

2. Locate the line indicating the user to be changed and substitute the application direction for the shell path as described above.

   [login name][all other fields]:[/bin/shell]

   [login name][all other fields]:[special application path]

3. When done, write and quit the file.

Note: the special application may create the user's directory automatically. However, if it does not, use **mkdir(1)** to make the needed directory in the appropriate file system.

The ONYX Office, developed by Onyx Systems, Inc., is an example of a special application for which a turnkey account would be created.

Managing the Print Queue

This section describes the procedures for two levels of printer scheduler management under the UNIX system. These two levels of management are:

- User printer request management
- Administrator printer request management

User Printer Request Management

This level involves the users' control over initiating and cancelling printer requests.

Note: the following procedures assume the parallel printer as the default destination.

PROCEDURE: **How to initiate a printer request**

1. Using the lp(1) command, a user may schedule a file for printing as follows:

   ```
   # lp [options] [file name]   <cr>

   request id is printer1-01 (# of files)
   #
   ```

2. To display the status of the request, do the following:

   ```
   # lpstat -t  <cr>

   scheduler is running
   system default destination: printer1
   member of class printer1:
         plp
   device for plp: /dev/plp
   plp accepting requests since [date]
   printer1 accepting requests since [date]
   printer plp now printing serial-01 enabled since [date]

   printer1-01   [owner]  [size of file]  [date] on slp
   #
   ```

If a printer request has been made in error, it can be cancelled as shown in the following procedure.

PROCEDURE: **How to cancel a printer request**

1. Determine the status of the request using lpstat as shown above, noting the file's "id."

2. To cancel the request, perform the following:

   ```
   # cancel printer1-01  <cr>

   request "printer1-01" cancelled
   #
   ```

   [Note: cancel instructs the system to stop printing the file immediately!]

Administrator Printer Request Management

This level allows the administrator to manage and control the scheduler and request mechanism through the use of six programs.

These six programs are:

- accept(1M) -- allows printer requests to be accepted.
- reject(1M) -- inhibits printer requests.
- lpstat(1) -- displays the status of printer requests.
- lpsched(1M) -- activates the printer scheduler.
- lpshut(1M) -- deactivates the printer scheduler.
- lpmove(1M) -- redirects printer requests to another destination (printer).

Note: to use these commands, the administrator must either log in as root or become a super-user. In addition, the "destination" printer assumed is **printer1**.

The **accept** command prepares and enables the printer facility to accept user's requests. The following procedure illustrates its use.

PROCEDURE: **How to use the accept command**

1. To enable the printer facility, perform the following:

   # /usr/lib/accept printer1 <cr>

   destinaton "printer1" now accepting requests
   #

The **reject** command inhibits and disables the printer facility from accepting any user requests. The following procedure illustrates its use.

PROCEDURE: **How to use the reject command**

1. To disable the printer facility, perform the following:

   ```
   # /usr/lib/reject -r["mesg"] printer1  <cr>

   destination "printer1" is no longer accepting requests
   ```

   [Note: the "-r" option allows the administrator to provide a "reason" message which is displayed each time a user attempts to request the disabled facility (e.g., "printer down for maintenance").]

   ```
   #
   ```

The **lpstat** command, previously described and illustrated, allows the administrator or a user to display the current status of print requests and their destinations (printers). In most cases, using the "-t" option (print all the status) is desirable; however, there are other options which can be invoked. These other options are described in the Enhanced ONYX System V USER REFERENCE MANUAL under the **lpstat(1)** section.

The **lpsched** command enables the printer scheduler facility for queuing requests. The following procedure shows its use.

PROCEDURE: **How to use the lpsched command**

1. To enable the scheduler, perform the following:

   ```
   # /usr/lib/lpsched  <cr>
   #
   ```

The **lpshut** command disables the printer scheduler facility from queuing requests. The following procedure shows its use.

PROCEDURE: **How to use the lpshut command**

1. To disable the scheduler, perform the following:

   ```
   # /usr/lib/lpshut  <cr>

   scheduler stopped
   #
   ```

   [Note: disabling the scheduler does not remove any requests currently queued. Therefore, when the scheduler is reenabled those queued requests are serviced.]

The **lpmove** command performs two tasks: first, it can move any or all requests currently queued to another destination (printer) or second, it can redirect all future requests from the current destination to another destination. The following procedures illustrate both forms of the command.

```
****************** CAUTION ************************
*                                                  *
* The lpmove command must not be invoked while     *
* the scheduler is enabled!  In addition, in its   *
* second form the prior destination (printer)      *
* is left disabled.                                *
*                                                  *
****************************************************
```

PROCEDURE: **How to move requests using the lpmove command**

1. Use the **lpstat** command to determine whether or not the destination (printer) is accepting requests. If not, activate it using **accept(1)**. In addition, note all the request id's to be moved.

2. Inform all users, using **wall(1)**, that the scheduler is being disabled and that the request id's are being moved to another destination (printer).

3. Disable the scheduler as follows:

   # /usr/lib/lpshut <cr>

   scheduler stopped
   #

4. To move the desired requests, do the following:

   # /usr/lib/lpmove [id's] [destination] <cr>

   total of [#] of requests moved to [destination]
   #

5. Reenable the scheduler as follows:

   # /usr/lib/lpsched <cr>
   #

6. Notify all users that the move is complete and that the scheduler is again enabled.

PROCEDURE: **How to redirect the destination using lpmove**

1. As described above, determine whether or not the new destination is active; if not, activate it. Then inform all users of the pending change.

2. Disable the scheduler as follows:

   # /usr/lib/lpshut <cr>

   scheduler stopped
   #

3. To redirect the destination, perform the following:

   # /usr/lib/lpmove [dest1] [dest2] <cr>

   destination [dest1] is accepting requests

   move in progress...

   total of [#] requests moved from [dest1] to [dest2]
   #

4. Reenable the scheduler as follows:

   # /usr/lib/lpsched <cr>
   #

5. Inform all users that the redirection is complete and the scheduler is reenabled.

Communications With Users

This section expands upon two utilities that have already been discussed: the **mail(1)** and **news(1)** commands, and introduces two more ways to talk to users: the **wall(1)** and **write(1)** commands.

The **mail(1)** command is basically a file editor with communications ability. Mail allows users to send and receive letters electronically. The procedure for sending mail is shown in the example below.

PROCEDURE: **How to send mail**

1. To send mail, do the following:

   $ mail [user name] <cr>

   Subject: [message]

   [Compose your letter; when done,
   do the following:]

   <cntrl-d>
   $

   [If sending mail via **uucp**, the
   additional query will appear:]

   Cc:[**system name to send to**] <cr>
   $

The **news**(1) command searches the directory /usr/news and displays the contents of all the files contained therein. Since all users have access to this directory, the system administrator can place whatever special information needs to be noted by users in a file in /usr/news.

To display news, a user enters: **news** <cr> on his/her terminal while logged onto the system.

The **wall**(1) command, an acronym for "write-all-users," can be used by the system administrator to notify all currently active users of some needed information immediately.

Note: Wall is a privileged command; therefore, the system administrator must either log in as root or become a super-user!

The procedure for writing to all users is as follows.

PROCEDURE: **How to send a message to all active users**

1. To send a message to all users, do the following:

   # /etc/wall <cr>

   [Enter the message; when done, enter
   the following:]

   <cntrl-d>

   Broadcast message from root...
   [The message]
   #

The system administrator can also inform users about necessary information by placing the message in the file called "motd" (message-of-the-day) in the /etc directory. **Motd** is displayed each time a user logs onto the system.

The **write(1)** command is used to establish an interactive conversation between users. The major anomaly associated with this command is that the user sending a message must flag the recipient when the message is complete. The old radio colloquialisms "over" and "out" have been widely used for this purpose; in the session they are represented by "-o-" for "over," and "-oo-" for "out."

PROCEDURE: **How to communicate a message to another user**

1. Determine the user's name and tty terminal number that you want to call. To do this, use the **who(1)** command as follows:

   $ who <cr>

   [user's name] [tty number] [date]
   $

2. Write the message to that user as follows:

   $ write [user name] [tty number] <cr>

   [Write the message. If a response is desired, enter **-o-**; otherwise, enter **-oo-**.]

   <cntrl-d>
   $

3. The "receiver" will get the message:

   Message from [your user name] [your tty] ...

   [message]

4. To respond, the "receiver" would perform the same sequence as described above.

Establishing Communication Links

This section describes the **cu** and **uucp** communication facilities previously discussed in Chapter 4, and illustrates their use for establishing communications between a local and remote system.

Note: prior to the invocation of either facility, all hardware elements such as modems or automatic calling units should be installed, tested, and ready for use. In addition, all pertinent system files should be configured as described in Chapter 4, under the sub-heading of "Setting Up Asynchronous Communications."

Using the cu Facility

The following procedure shows two ways to achieve communications between a local and remote computer system using **cu(1C)**.

PROCEDURE: **How to establish a communications link with cu**

Example 1:

```
$ cu -s1200 -ltty[number] [phone number] <cr>
Connected

[ Where "-s" denotes the baud rate and "-l" the
  port number being used. ]

login: [from the remote computer]
```

Example 2:

```
$ cu -s1200 -ltty[port number] dir  <cr>

[ "dir" means a direct dedicated telephone line.
  If talking directly to a modem, the modem
  must be initialized and dialed manually. ]

Connected
login: [from the remote computer]
```

To execute commands or terminate communications under **cu**, the command must be prefixed by a **tilde** (~) character.

For example, to log off enter: **~. <cr>**, instead of **<cntrl-d>** the normal sequence.

It should be noted that the data integrity of files transferred using **cu** is not verified automatically. Therefore, the system administrator must perform the verification manually using the **sum(1)** command. This command reads the contents of a file and uses a mathematical algorithm to produce a unique number referred to as a "checksum." This checksum number is used to determine whether or not two files are identical.

The following procedure illustrates the use of **sum(1)**.

PROCEDURE: **How to use the sum command**

1. Perform a file checksum as follows:

```
# sum [file name]  <cr>

[checksum number] [blocks] [file name]
#
```

The system administrator should perform a checksum for any files transferred using **cu(1C)**.

The following procedure shows how to transfer files between two computers that have established a communications link using **cu(1C)**.

PROCEDURE: **How to transfer files with cu**

Example 1:

Transfer a file from YOUR computer to THEIRS.

```
$ ~%put [/directory/myfile][/directory/theirfile] <cr>

[ Displays the number of lines, characters, and
  elapsed time for the file being transferred. ]
$
```

Example 2:

Transfer it from THEIR computer to YOURS.

```
$ ~%take [/directory/theirfile][/directory/myfile] <cr>

[ Displays the number of lines, characters and
  elapsed time for the file being transferred. ]
$
```

Using the uucp Facility

The **uucp(1C)** utility is primarily a controlled UNIX-to-UNIX file transfer facility. Its "control" relates to the management, auditing, and checksumming activities performed during each session. Therefore, it is preferred over **cu** for file transfer.

This discussion and examples focus on the basic files involved and how to invoke a simple session.

There are seven files, in addition to those previously discussed, that are of concern to the system administrator for establishing communications.

They are:

- AUDIT
- LOGFILE
- USERFILE
- STST
- ERRLOG
- SYSLOG
- LCK..

All of these files, except USERFILE, reside in /usr/spool/uucp. USERFILE resides in /usr/lib/uucp.

The **AUDIT** file contains some of the session activity associated with a remote computer (Master) attempting to access a local (Slave) computer. Under **uucp**, a computer may assume the role of either a "Master" or "Slave" depending upon the activity that is undertaken.

The **ERRLOG** file contains the errors associated with any given **uucp** session.

The **LOGFILE** file contains the complete history of events associated with any given **uucp** session.

The **SYSLOG** file contains an extrapolated version of the LOGFILE, of just those entries pertaining to the master's activity, and it provides user accounts data and specific data on the size of the file(s) transferred.

The **USERFILE** file contains a list of directories and their pathnames to which a calling computer is restricted.

The **LCK..** file inhibits the initiating of **uucp**. Therefore, if this file exists, it must be removed prior to invoking a session.

The **STST** file contains data relating to the system's attempt in retrying a call that previously failed to a remote computer.

Note: the system is already configured with a **uucp** login account where a user invokes **uucp**. In addition, all files transferred during a session will have **uucp** as their owner. Therefore, it may be necessary to change the file ownership after the file has been received.

The following procedures show how to send and receive a file between computers under **uucp**. For clarity, "myfile" is the file being sent to the remote computer, "them" is the remote account name, "theirsys" is the remote computer's **uucp** name, and "theirfile" is a file on the remote computer.

PROCEDURE: **How to send a file using uucp**

1. Send a file to a remote system as follows:

```
$ uucp -m -nthem myfile theirsys!~them <cr>
$
```

[Where:
-m = notify me by mail
-n = notify them by mail]

PROCEDURE: **How to copy a file using uucp**

1. Copy a file from the remote system as follows:

```
$ uucp -m theirsys!/dir/theirfile /dir/myfile <cr>
$
```

System Accounting

This section describes how to use the system accounting facility in the UNIX system.

The accounting function is initialized only when the appropriate commands are incorporated into the /etc/rc file. If initialized, accounting compiles statistical data about system operation on a daily basis while the system is in the multi-user mode. All of the data gathered is placed in files for later viewing.

Each day, the system administrator should print these files on the printer, then purge them so they do not consume valuable disk storage space. The procedure to do this is as follows.

PROCEDURE: How to manage system accounting

1. Change directories to /usr/lib/acct/.

```
# cd /usr/lib/acct  <cr>
#
```

2. Print the accounting files using "prdaily" and save them in /usr/adm/acct/sum/report.

```
# prdaily>/usr/adm/acct/sum/report <cr>
#
```

3. Change directories to /usr/adm/acct/sum and print the report file onto a printer.

```
# cd /usr/adm/acct/sum  <cr>
#

# lp report   <cr>
#
```

4. Look at the contents of the /usr/adm/pacct file. If desired, print the contents onto a printer.

5. Purge the files as follows:

```
# cat /dev/null >/usr/adm/pacct <cr>
#

# cat /dev/null >report   <cr>
#
```

6. All reports saved should become part of the system administrator log.

System Performance Considerations

At various times the system may appear slow in completing a requested task. This may be the result of a variety of conditions; however, there is a known set of conditions which contributes to performance degradation.

Among these are:

- Insufficient disk storage
- A poorly organized file system
- Insufficient memory
- An excessive amount of background process activity
- An unmanaged accounting facility
- An inordinate amount of user interaction
- An unmanaged printer spooler facility
- An unterminated terminal cable

Elements such as insufficient disk storage and insufficient memory may be caused by reaching the physical limitations of these specific devices.

Note: part of the system administrator's job is to determine when physical system expansion is necessary.

The following examines each of the topics above and defines what a system administrator can do to maintain a reasonable level of system performance.

Insufficient disk storage should be checked for daily with the **df(1)** and **du(1)** commands. Df displays the total number of blocks used and how many are still available within a given file system. Du further isolates storage usage by listing the blocks used by each user directory and file within a given file system.

Note: if a file system shows less than 100 blocks available, the system administrator must go through the system and remove or purge all unnecessary user files and temporary system files.

A poorly organized file system causes the disk drive to work "overtime" when attempting to reassemble a file. To minimize this condition the system administrator can use the **cpio(1)** command to write user file systems to tape, remake the file system using **mkfs(1)**, and write them back onto disk contiguously. In addition, the **fsck(1M)** program can restructure a file system's "free list," making it more compact. All of these procedures are discussed individually in Chapter 4, "Getting Started."

An excessive number of background processes causes the system to "swap-out" processes extensively. Swapping out a process consumes a considerable amount of system time. The system administrator can check what processes are active on the system with the **ps(1)** command.

Sometimes processes become "orphaned," meaning that they are not removed from the process table after completion. When the number of orphaned processes becomes large, it can inhibit other processes from being acted upon. The **kill(1)** command is used to terminate a process manually.

An unmanaged accounting facility consumes large amounts of disk storage space because it constantly accumulates statistical data about the system. The applicable accounting files should be purged daily to avoid this situation.

An inordinate amount of user activity means, for this purpose, users who indiscriminately invoke processes that use large amounts of system memory. For example, some of the "games" which may be supplied with the system may fall into this category. Some of these games can be played for hours without conclusion, and thereby can significantly reduce the amount of system memory available for necessary tasks.

An unmanaged printer spooler can result in an unnecessary amount of background processes being generated. Each time a file is queued to be printed, a background process is established. When spooled files are not removed after printing or the same file is queued more than once, valuable system space is taken up. These files are removed using **rm(1)** and their processes terminated using **kill(1)** commands.

Last, an unterminated terminal cable means that one end of the cable has been connected to the system but the other end remains free. By not connecting (terminating) the free end, the cable acts as an "antenna" which can cause erroneous signals to be sent to the system. This could result in unpredictable system operation.

Handling Users Problems

Many problems occur when a user is learning UNIX. These can be avoided by providing a basic instructional plan for new users. Therefore, this discussion applies to conceptual rather than actual user problems.

When problems occur, the system administrator should:

1. Gather all the relevant information from the user as follows:

   (a) What was done prior to the problem
   (b) What was done after the problem
   (c) Any error messages presented
   (d) All symptoms exhibited

2. Check the system process(es) status associated with this user.

3. Check the administrator's log to see if the problem has occurred before.

4. Analyze all information gathered **before** taking any action.

5. Take the necessary corrective action indicated by the problem analysis.

6. Write the problem and solution in the system administrator's log.

A system administrator may encounter the following problems:

- Improper exiting from an edit session
- Too many background processes running
- Not enough disk storage space
- Runaway processes
- A user who can no longer interact with the system, sometimes called a "hung" terminal

Correcting these problems may involve copying or rewriting files, killing processes, removing files, restarting processes, and reconfiguring file parameters.

Handling Errors

In the UNIX system, error messages can originate from the kernel itself, or from any one of the vast number of utility programs available. However, error messages are categorized as follows:

- Those which inform a user of a destructive condition
- Those which warn a user of a potential problem
- Those which are of an instructional nature

Any error message that is preceded by the caveat **panic:** indicates that a condition has arisen in the system that, if allowed to continue, would ultimately result in the destruction of data. Therefore, when such a condition occurs, the UNIX operating system ceases operation immediately and informs the system administrator on the console terminal.

The next category includes those messages the system issues to warn of a potential problem. Two such messages might be: "no space" left on a file system and "no more processes," indicating the system process table is full.

The last category includes those messages which inform a user that the system could not interpret the user's request. One such message might be "syntax error," indicating that a command was not entered in the proper format.

In handling errors, the most important action is to write down all the error information displayed and the action that produced the error! In the event additional help from the dealer's technical support personnel must be sought, these people cannot be of assistance unless this information can be related exactly.

An Approach

One approach the system administrator can follow for handling errors is:

1. Write down all error information provided.
2. If possible, determine what action occurred just prior to the generation of the error message.
3. Check the log to see if it has occurred before.
4. Attempt to define and interpret the meaning of the error message.
5. Attempt to isolate its origin.

6. Effect a solution, or call for help.

7. Record the error and solution, when known.

```
********************** CAUTION ***************************
*                                                        *
* For panic: error conditions, the system must always    *
* be rebooted.  In doing so, always perform an fsck to   *
* check for errors.                                      *
*                                                        *
**********************************************************
```

Tools

Some of the tools a system administrator uses in isolating an error message's origin and determining a possible cause are:

- df(1), du(1) -- checks disk storage space
- fsck(1M) -- verifies file system integrity
- grep(1) -- matches patterns globally
- strings(1) -- displays the ASCII strings in an executable program
- od(1) -- provides an octal dump of a file allowing you to look for non-printable characters
- ncheck(1M) -- locates an owner of a file file through its i-node number.
- find(1) -- finds out where a file resides in the system

The first two commands have already been described. The remaining five are described below with examples.

The **grep(1)** command scans a file to match the variable parameter supplied. Therefore, use **grep** to find out if an error message originated from the suspected command.

PROCEDURE: **How to use the grep command**

1. Become super-user and perform the following.

```
# grep '[error msg]' [/directory][/command] <cr>

[ Matches, if there, or exits ]
#
```

The **strings(1)** command displays all the ASCII format messages in an executable (command) type file.

PROCEDURE: **How to use the strings command**

1. Become super-user and perform the following.

```
# strings [directory][command] |more  <cr>

[ Displays all ASCII messages. ]
#

[ Using the more(1) utility in conjunction
  with the command string above allows the data
  to be viewed one page at a time. ]
```

The **od(1)** command displays a file in its octal and ASCII form, so that a comparison of the data to the ASCII table located in Section 7 of the Enhanced ONYX System V USER REFERENCE MANUAL is possible. This is helpful in looking for non-printable characters that may have been entered inadvertently.

PROCEDURE: **How to use the od command**

1. Look for non-printable characters.

```
# od -bc [/directory] [/file name] |more  <cr>

0000000 007  S   Y   S   T   E   M
        007 123 131 123 124 105 115  040

#

[ Where:

  0000000 = the byte count field in Octal.
            Top line = ASCII representation.
            Bottom line = octal representation.
      040 = a space character.
      007 = the non-printable character, in this
            case a <control-g>. ]
```

The **ncheck(1M)** command is used when output from **fsck** determines a damaged file, but only gives the i-node number and not the file's name.

For the example below, assume the i-node number is 10 in file system /dev/dsk/c0d0s3.

PROCEDURE: **How to use the ncheck command**

1. Become super-user and do the following:

```
# /etc/ncheck -i 10 /dev/dsk/c0d0s3   <cr>

/dev/dsk/c0d0s3:
10      /[directory/file name]
#
```

The **find**(1) command traverses the directory hierarchy of pathnames in an attempt to locate and display the path for the desired file.

PROCEDURE: **How to use the find command**

1. Display a file's path as follows:

```
# find / -name [file name] -print <cr>

[/file system/directory/sub-directory/file name]
#
```

[Note: the "/" instructs the system to start at root and move downward through the pathname tree looking for every occurrence of the named file.]

Further information on what can be done when problems arise is contained in Chapter 7, "Handling System Problems."

Summary

This chapter discusses the tasks perceived to be a part of a daily routine for the system administrator.

This chapter includes the following topics:

- System startup and shutdown
- Backing up and restoring data
- Managing processes
- Communicating with users
- Establishing communication links
- System disk management
- Adding and removing data
- Creating user accounts
- Handling problems
- System performance management

In conjunction with these topics, the following procedures are described:

- How to initialize the system
- How to use the shutdown program
- How to determine disk storage usage
- How to further isolate disk storage usage
- How to purge data files
- How to use the null device to purge files
- How to rebuild the system free list
- How to back up a file system
- How to restore a file system
- How to save a directory using cpio
- How to restore a directory using cpio
- How to save a single file using cpio
- How to write and retrieve a file on a tape track
- How to create and remove a directory
- How to create a null file
- How to copy a file

- How to move a file
- How to remove a file
- How to change a file's permissions
- How to change a file's ownership
- How to change the group ownership of a file
- How to kill a process
- How to kill all processes simultaneously
- How to use the adduser command
- How to add user accounts manually
- How to create a turnkey account
- How to initiate a printer request
- How to cancel a printer request
- How to use the accept command
- How to use the reject command
- How to use the lpsched command
- How to use the lpshut command
- How to move requests using the lpmove command
- How to redirect the destination using lpmove
- How to send mail
- How to send a message to all active users
- How to communicate a message to another user
- How to establish a communications link with cu
- How to use the sum command
- How to transfer files with cu
- How to send a file using uucp
- How to copy a file using uucp

- How to manage system accounting
- How to use the grep command
- How to use the strings command
- How to use the od command
- How to use the ncheck command
- How to use the find command

TABLE OF CONTENTS

CHAPTER 6

CHAPTER 6

SYSTEM EXPANSION

Preface

The goal of this chapter is to instruct the system administrator on the procedures for adding serial printers and expansion disk drives.

Note: information on making changes to the parallel printer is described in Chapter 4, "Getting Started," under the heading of "Configuring Printers."

Remember, while following a procedure all commands and actions requested appear in **bold type**. Any system response to a requested command will appear immediately below that command. In addition, all comments made within a procedure are contained between brackets "[]." The symbol <cr> means "carriage return."

Adding More Serial Printers

This section shows how to incorporate and configure the system to support a second serial printer.

The procedure includes the following steps:

1. Altering parameters in /etc/inittab
2. Configuring the spooler facility

Note: while performing this procedure, the administrator must either log in as root or become a super-user. In addition, the printer scheduler must be turned off!

In this example procedure, port "tty02" is used to support the added serial printer. The printer class name assigned is "printer2;" which assumes that at least one other printer currently exists. If more than two printers are currently configured, than rename as desired. The printer class name is user definable.

PROCEDURE: **Modifying /etc/inittab**

1. Edit the desired tty entries in the inittab file for the init 2 level.

   ```
   02:2:respawn:/etc/getty tty02 9600155
   02:2:off:/etc/getty tty02 9600155
   ```

2. Write and quit the /etc/inittab file.

PROCEDURE: **How to configure the spooler facility**

1. Change directories to /usr/lib.

   ```
   # cd /usr/lib  <cr>
   #
   ```

2. Deactivate the printer scheduler as follows: desired device parameters.

   ```
   # lpshut  <cr>

   scheduler stopped
   #
   ```

3. Added to serial printer as follows:

   ```
   # lpadmin -ptty02 -cprinter2 -mdumb -v/dev/tty02  <cr>
   ```

4. Verify the added printer is configured by performing the following:

   ```
   # lpstat -t  <cr>

   scheduler is not running
   system default destination: printer1
   members of class printer1:
           slp
   members of class printer2:
           tty02
   device for slp: /dev/slp
   device for tty02: /dev/tty02
   slp accepting requests since [date]
   printer1 accepting requests since [date]
   tty02 not accepting requests since [date]
           new destination
   printer2 not accepting requests since [date]
           new destination
   printer slp is idle.  enabled since [date]
   printer tty02 disabled since [date]
           new printer
   #
   ```

5. Reactivate the printer scheduler as follows:

```
# lpsched  <cr>
#
```

Requests to print files can now be directed to either or both serial printers.

Adding More Disk Drives

Adding another disk drive involves five basic tasks:

- Formatting the drive
- Configuring the drive
- Making the special device file
- Making the file system
- Labelling the file system

Formatting the Drive

Formatting means preparing the drive's basic logical layout. A disk drive is formatted using the **format(1M/1SA)** program; one of the utilities contained in the standalone directory.

The procedure below illustrates formatting the second drive (drive 1) a 40 megabyte drive (r203e), with an Adaptec disk controller and preparing the drive's total storage capacity to be configured as a single file system.

Note: **format** is a standalone program, therefore, the system must be in the standalone environment. In addition, the Onyx 6810 deskside and desk top systems use the **Adaptec5500** controller.

```
********** WARNING **********
*                           *
* The format program destroys ALL data on the named *
* drive and once invoked it cannot be stopped!      *
* Therefore, ensure the requested drive to be       *
* formatted is the correct one!                     *
*                           *
*****************************
```

PROCEDURE: How to use the format program

1. From the standalone shell, perform the following:

```
SHELL$$ format -c 0 -d 1 -s r203e -t adaptec55 -l DRIVE1 <cr>
Loading /stand/format.

[ Where:
        -c 0 = the controller bus address.
        -d 1 = the physical drive number.
    -c r203e = the model number of the drive.
-t adaptec55 = the type of disk controller used.
   -l DRIVE1 = an optional name to be recorded
               in sector zero. ]

format: initial format with no data, type adaptec, size r203e

format: formatting disk.

format: checking entire drive for bad sectors;

    number of sectors 34560

checking cylinder [number]

[ Program advances and checks each cylinder until
  it reaches the total number of sectors. ]

format: writing out sector zero
SHELL$$
```

Note: if errors occur during the format process, write down the error data and contact your dealer's technical support personnel for assistance!

Configuring the Drive

The second phase involves partitioning (slicing) the disk's storage space into defined logical file systems and building a "map" reflecting these file systems, their locations and sizes. This "slicing" process is accomplished using the **diskconf(1M/1SA)** program, another utility executed only in the standalone environment.

The following procedure shows how to use the **diskconf** program to reserve five blocks of storage for bad sector sparing and allocate the remaining disk storage for the user's workspace.

Note: like **format**, **diskconf** must be used in the standalone environment.

PROCEDURE: **How to use the diskconf program**

1. From the standalone shell, perform the following:

```
SHELL$$ diskconf -c 0 -d 1 -s 5 0 -b  <cr>
Loading /stand/diskconf.

[ Where:
      -c 0 = the controller bus address.
      -d 1 = the physical drive number.
    -s 5 0 = the size, in sectors, of the
             file system(s) to be made.
        -b = the "boot" flag which configures
             the disk for booting.

diskconf: new slice structure - number of slices 2
    slice 0, offset 0, size 5
    slice 1, offset 5, size 34555
    slice 2, offset 0, size 0
    slice 3, offest 0, size 0
    slice 4, offset 0, size 0
    slice 5, offset 0, size 0
    slice 6, offset 0, size 0
    slice 7, offset 0, size 0

SHELL$$

[ Note: slice 0 (file system c0d1s0) reserves
  five blocks (1024 bytes each) for spared
  sectors and slice 1 (file system c0d0s1)
  consumes the remaining available disk
  storage. ]
```

Creating the Device File

The third phase of this process involves creating the device file "node" entries in the /dev/dsk and /dev/rdsk directories for this added disk drive. These "nodes" are created automatically using the **devices** command in the /etc directory. The **devices** program assigns the appropriate device names, mode types (blocked and character), and the major and minor device numbers for the added device.

Devices builds all the nodes to its maximum configurable number (e.g., c0d1s0 through c0d1s7) whether or not they currently exist on the system. This affords the administrator the flexibility to reconfigure, alter, or add file systems to a disk without having to make their "nodes" manually. **Devices** is also used to add other peripheral device nodes such as another cartridge tape drive.

Note: the **devices** program is performed in the single-user environment and requires the devices (such as disk, cartridge or nine track tape) to be turned on.

PROCEDURE: **How to use the devices program**

1. In single-user mode, perform the following:

```
# /etc/devices -w  <cr>
#
```

2. Display the configuration structure map as follows:

```
# /etc/devices  <cr>

                The scsi config struct

contrl unit 0 unit 1 unit 2 unit 3 unit 4 unit 5 unit 6 unit 7
 00000  DISK   DISK
 00001  CART
 00002
 00003
 00004
 00005
 00006
 00007

[ Display indicates all currently configured and
  physically existing devices. ]
#
```

3. Verify the nodes exist in the /dev/dsk and /dev/rdsk directories as follows:

```
# ls -l /dev/dsk/c0d1s1  /dev/rdsk/c0d1s1 <cr>

br--------- 1 root system 0, 9 [date/time]/dev/dsk/c0d1s1
cr--------- 1 root system 4, 9 [date/time]/dev/rdsk/c0d1s1
#
```

Making the File System

The fourth phase involves making the actual logical file system through the use of the **mkfs(1)** program.

```
********** WARNING **************
*                               *
* This procedure must be done in *
* the single-user environment!   *
*                               *
*********************************
```

PROCEDURE: **How to make the file system**

1. Make the file system as follows:

```
# mkfs /dev/dsk/c0d1s1  <cr>

bytes per logical block = 1024
total logical blocks = 34555
total inodes = 8624
gap (physical blocks) = 1
cylinder size (physical blocks) = 1
#
```

Labelling the File System

The fifth phase is to label (name) the newly created file system. This is done using the **labelit** command. In the example below, **NEWFS** was selected for the file system name and **DRIVE1** for the volume name; however, they are user definable.

PROCEDURE: **How to use the labelit command**

1. To label the file system, perform the following:

```
# labelit /dev/dsk/c0d1s1 NEWFS DRIVE1  <cr>

Current fsname:[blank], Current volname:[blank], Blocks: 69106,
Inodes: 8592, FS Unit: 1Kb, Date last mounted:[date],
NEW fsname = NEWFS, NEW volname = DRIVE1 -- DEL if wrong !!
#
```

Summary

This chapter discusses how to add more serial printers and disk drives. In addition, this chapter provides an example of the following procedures:

- Modifying /etc/inittab
- How to configure the spooler facility
- How to use the format program
- How to use the diskconf program
- How to use the devices program
- How to make the file system
- How to use the labelit command

TABLE OF CONTENTS

CHAPTER 7

LIST OF ILLUSTRATIONS

CHAPTER 7

HANDLING SYSTEM PROBLEMS

Preface

This chapter describes the tools for defining, verifying, and isolating system problems.

The topics in this chapter include:

- System error reporting
- FSCK diagnostic messages
- Sparing bad disk sectors
- System crash procedures
- Restoring the system after a crash
- Emergency shutdown
- Who to call for help

Remember, while following a procedure all commands and actions requested appear in **bold type.** Any system response to a requested command appears immediately below that command. In addition, all comments made within a procedure are contained between brackets "[]." The symbol <cr> means "carriage return."

System Error Reporting

This section discusses the various classes of system errors which might occur and directs the system administrator to those manuals where such error information is detailed.

Generally, system errors are divided into two major classifications:

- Primary bootup (hardware/software) errors
- UNIX operation (hardware/software) errors

Primary Bootup Errors

Primary bootup errors are those which occur during the system **self test** (hardware) phase or during the initial attempt to boot the UNIX system software. A failure at this level prevents the system from booting up and generally indicates that either a hardware component has failed or that the system software is severely corrupted.

The list of **self test** errors and their descriptions is found under the "Troubleshooting" section of the ONYX 6810 MICROCOMPUTER SYSTEM USER'S GUIDE that accompanies the system.

UNIX Operation Errors

In the UNIX system, error messages can originate from the kernel itself, or from any one of the vast number of utility programs available. However, this class of errors is categorized further as follows:

- Those which inform a user of a destructive condition
- Those which warn a user of a potential problem
- Those which are of an instructional nature

Any error message that is preceded by the caveat **panic:** indicates that a condition has arisen in the system that, if allowed to continue, would ultimately result in the destruction of data. Therefore, when such a condition occurs, the UNIX operating system ceases operation immediately and informs the system administrator on the console terminal.

The next category includes those messages the system issues to warn of a potential problem. Two such messages might be: "no space" left on a file system and "no more processes," indicating the system process table is full.

The last category includes those messages which inform a user that the system could not interpret the user's request. One such message might be "syntax error," indicating that a command was not entered in the proper format.

In handling errors, the most important action is to write down all the error information displayed and the action that produced the error! In the event additional help from the dealer's technical support personnel must be sought, these people cannot be of assistance unless this information can be related exactly.

The list of these types of errors is found in the section entitled: **intro(2)**, in the Enhanced ONYX System V PROGRAMMER REFERENCE MANUAL.

System Error Logging

As an aid in determining system integrity, the UNIX system records and maintains a log of certain types of errors which may occur during system operation. The error types of concern are as follows:

- Stray interrupts
- Memory parity errors
- Thermostat interrupts
- SCSI bus errors

Stray interrupt errors occur when the system detects an interrupt signal but is unable to determine its origin. This type of error may occur when the system attempts to service an inordinate amount of requests (such as multiple user terminal I/O) simultaneously.

Memory parity errors occur when data is corrupted to such an extent that the system memory error correction hardware cannot correct it. Errors of this type may indicate a badly damaged data file or a failure in the system memory subsystem itself.

Thermostat interrupt errors occur when the system hardware detects that the ambient temperature exceeds the safe operating range for the equipment (approximately 85 degrees fahrenheit); therefore, causing the system to cease operation to avoid component damage.

SCSI errors refer to the failure of a peripheral device (such as the disk or tape drive) to complete a requested task. Errors of this type generally apply to reading and/or writing data unsuccesfully between the system and the device.

The system administrator displays the accummulated error data using the **errpt(1M)** command. The actual error data is contained in **errfile** which resides in the /usr/adm directory. The following procedure describes the use of **errpt**.

PROCEDURE: **How to use the errpt command**

1. Perform the command as follows:

```
# errpt  <cr>

Summary Error Report prepared on [date]  page 1

Error Types: all
Limitations:
Date of Earlist Entry: [date]
Date of Latest Entry:  [date]

Total Stray Interrupts          - [number]
Total Memory Parity Errors      - [number]
Total Thermostat Interrupts     - [number]
Total SCSI Errors               - [number]

DISK Controller 0 Unit 0

Hard Errors                     - [number]
Soft Errors                     - [number]
Total I/O Operations            - [number]
Total Misc. Operations          - [number]
Errors Missed                   - [number]
Total SCSI Errors               - [number]

CARTRIDGE TAPE Controller 1 Unit 0

Hard Errors                     - [number]
Soft Errors                     - [number]
Total I/O Operations            - [number]
Total Misc. Operations          - [number]
Errors Missed                   - [number]
Total SCSI Errors               - [number]

#
```

The **errpt(1M)** command has two significant options: the **-d** argument which allows the administrator to search and display just that error data pertaining to a specific named device such as disk file system "c0d0s0" and the **-a** argument which displays the entire detailed version of the log. Because of its detail, the **-a** argument is particularly helpful for determining the actual type of failure on a peripheral device.

Note: the error logging facility in UNIX does not purge (flush-out) "errfile" automatically. Therefore, the administrator should program the system to periodically printout and then purge /usr/adm/errfile. A suggested procedure is described in section 1M of the Enhanced ONYX SYSTEM V ADMINISTRATOR GUIDE VOLUME II.

An Approach

One approach the system administrator can follow for handling errors is:

1. Write down all error information provided.
2. If possible, determine what action occurred just prior to the generation of the error message.
3. Check the log to see if it has occurred before.
4. Attempt to define and interpret the meaning of the error message.
5. Attempt to isolate its origin.
6. Effect a solution, or call for help.
7. Record the error and solution, when known.

```
********************** CAUTION ***************************
*                                                        *
* For panic: error conditions, the system must always    *
* be rebooted.  In doing so, always perform an fsck to   *
* check for errors.                                      *
*                                                        *
**********************************************************
```

FSCK Diagnostic Messages

This section describes the various messages that the **fsck(1M)** program can produce upon detecting an error.

The text below groups these error messages in reference to the particular test phase that is invoked.

INITIALIZATION PHASE

Cannot fstat standard input: attempt to get status from the interacting terminal failed.

Cannot get memory: a request to place its tables in memory failed.

Cannot stat root: request for statistics from the "root" directory failed.

Cannot stat F: request for statistics about the named file system failed.

Cannot open F: the named file system could not be opened.

Cannot create F: request to create a scratch file failed.

CANNOT SEEK:BLK B: request to move to a specific file system block failed.

CANNOT READ:BLK B: request to read a specific block of a file system failed.

CANNOT WRITE:BLK B: request to write a specific block in a file system failed.

```
************** WARNING ****************
*                                      *
* These messages indicate extensive    *
* file system corruption! Therefore,   *
* rebuild the file system.             *
*                                      *
****************************************
```

PHASE 1: CHECK BLOCKS AND SIZES

UNKNOWN FILE TYPE I=I (CLEAR): the information in the i-node is not interpretable. If CLEAR is answered "y," an UNALLOCATED I-NODE message will appear.

LINK COUNT TABLE OVERFLOW: the i-node table used by **fsck** is not large enough to accommodate all the listed i-nodes.

B BAD I=I: the block number B contains an i-node with a lower number than the first block on the disk.

EXCESSIVE BAD BLKS I=I: there are many of these bad i-node numbers.

BAD/DUP I=I: an i-node contains a block number already claimed by another i-node.

EXCESSIVE DUP BLKS I=I: there are many i-nodes that contain the same block number.

DUP TABLE OVERFLOW: there are more of these i-nodes than can be accommodated by the DUP table.

POSSIBLE FILE SIZE ERROR I=I: the i-node size does not match the actual number of blocks used by that i-node.

DIRECTORY MISSALIGNED I=I: the size of a directory i-node is not a multiple of the size of the directory entries.

PARTIALLY ALLOCATED INODE I=I: the i-node number is improper.

Note: these messages indicate that minimal file system corruption has occurred. Therefore, a complete rebuild is not necessary.

PHASE 2: CHECK PATHNAMES

ROOT INODE UNALLOCATED. TERMINATING: the "root" i-node number is not correct.

ROOT INODE NOT A DIRECTORY (FIX): the i-node for "root" does not reflect that of a directory. If FIX is answered "y," this condition is corrected.

DUPS/BAD IN ROOT INODE: an i-node in the "root" directory has a duplicate block number.

I OUT OF RANGE I=I NAME=F (REMOVE): a directory has a file with an i-node number greater than the end of the list. If REMOVE is answered "y," the i-node is removed.

UNALLOCATED I=I OWNER=O MODE=M SIZE=S MTIME=T NAME=F (REMOVE): a directory has an i-node number that does not fit any known file. If REMOVE is answered "y," the i-node is removed.

DUP/BAD I=I OWNER=O MODE=M SIZE=S MTIME=T DIR=F (REMOVE): bad or duplicate blocks were found in a directory. **DUP/BAD I=I OWNER=O MODE=M SIZE=S MTIME=T FILE=F (REMOVE):** a file has a bad or duplicate block number. If REMOVE is answered "y," the file is removed.

BAD BLK BIN DIR I=I OWNER=O MODE=M SIZE=S MTIME=T: a bad block exists in the /bin directory.

```
************************* CAUTION ******************************
*                                                              *
* If there is an inordinate amount of unallocated/duplicate    *
* blocks, i-nodes, or files, then rebuild the file system!     *
*                                                              *
****************************************************************
```

PHASE 3: CHECK CONNECTIVITY

UNREF DIR I=I OWNER=O MODE=M SIZE=S MTIME=T (RECONNECT): a directory i-node was not connected to a directory. If the response to RECONNECT is "y," then it will reconnect it; otherwise, it will stay in the "lost+found" directory.

SORRY. NO lost+found DIRECTORY: the "lost+found" directory cannot be found.

SORRY. NO SPACE IN lost+found DIRECTORY: there is no room to accommodate another file.

DIR I=I1 CONNECTED. PARENT WAS I=I2: indicates that a directory was placed in the "lost+found" directory.

Note: these messages indicate that minimal file system corruption has occurred. No file system rebuild is necessary.

PHASE 4: CHECK REFERENCE COUNT

UNREF FILE I=I OWNER=O MODE=M SIZE=S MTIME=T (RECONNECT): same as the directory UNREF, except for a file.

LINK COUNT FILE I=I OWNER=O MODE=M SIZE=S MTIME=T COUNT=X SHOULD BE (ADJUST): the link count for an i-node is wrong. If ADJUST is answered "y," then the link count is adjusted.

FREE INODE COUNT WRONG IN SUPERBLK (FIX): this indicates the actual count of the free i-nodes does not match the count in the SUPERBLK. If FIX is answered "y," then the i-node count is corrected.

```
************************* CAUTION ******************************
*                                                              *
* If there is an inordinate amount of unreferenced files or    *
* incorrect link counts, then rebuild the file system!         *
*                                                              *
****************************************************************
```

PHASE 5: CHECK FREE LIST

EXCESSIVE DUP BLKS IN FREE LIST: indicates that the free block list contains more than a tolerable number.

BAD FREEBLK COUNT: indicates that the free block count is greater than 50 or less than 0.

X BAD BLKS IN FREE LIST: indicates the number of bad blocks in free list.

X DUP BLKS IN FREE LIST: indicates the number of duplicate blocks in the free list.

X BLK(S) MISSING: indicates that a number of blocks unused by the file system were not in the free list.

FREE BLK COUNT WRONG IN SUPERBLOCK (FIX): the actual count of free blocks does not match the number in the SUPERBLK. If FIX is answered "y," then the block count is corrected.

BAD FREE LIST (SALVAGE): bad blocks found in the free list. If SALVAGE is answered "y," then the following actions occur:

PHASE 6: SALVAGE FREE LIST: self explanatory.

CLEANUP X files Y blocks Z free: self explanatory.

**** BOOT UNIX (NO SYNC!) ****: type nothing further; press the reset button!

***** FILE SYSTEM WAS MODIFIED *****: self explanatory.

Note: some of the messages shown above actually appear in a single line on the terminal screen.

Sparing Bad Disk Sectors

The Onyx enhanced UNIX system V has two utilities pertaining to recording and sparing disk sectors. These two utilities are: **spare(8)** which allows the sparing of a bad disk sector and **sparelist(8)** which maintains the list of spared disk sectors.

The **sparelist** command displays the base sector, the total number of sectors in a slice, the number of alternate (spare) sectors allocated, and the sectors that have been previously spared. The following procedure illustrates displaying the spared sector information for device "c0d0s3."

Note: the **sparelist** command must be performed standalone mode.

PROCEDURE: **How to use the sparelist command**

1. In standalone mode, perform the following:

```
SHELL$$ sparelist /dev/dsk/c0d0s3  <cr>

/dev/dsk/c0d0s3 is a mounted file system
Spare information for Controller 0 Unit 0 Slice 3

Base sector            [number]
Number of sectors      [number]
Number of Alternates   [number]
Number of bad sectors  [number]

[ If sectors have been spared previously, the
  following additional information will appear: ]

Sector [number] -> Physical [number]

[ Otherwise the program displays... ]

There are no spared sectors on this slice
#
```

In addition, the administrator should periodically print this list onto a printer and incorporate it into the Administrator's Log for referrence.

The need to spare a sector is generally indicated by either consistent failures being logged and displayed by **errpt**, applicable errors noted by the **fsck(1M)** program or the immediate issuance of an error message to the terminal. As an example, assume that the following error message was displayed on the terminal:

```
disk error class 0x1, code: 0x0 block
number in slice: 454, Absolute block number: 9674
Uncorrectable data error on Disk 0, Drive 0, Slice 2
```

The following procedure shows how to spare sector 454 in slice (file system) 2.

Note: the **spare** command must be perfomed in the standalone shell environment!

PROCEDURE: **How to use the spare command**

1. In the standalone shell environment, perform the following:

   SHELL$$ **spare -c0 -d0 -s2 454** <cr>

   Sector to be spared: 454

   [Unless an error occurs during the sparing process, the action is silent.]

   SHELL$$

2. Use the **sparelist** utility to verify the bad sector is spared.

3. Reboot the system and initiate multi-user mode.

Note: an inordinate amount of bad sectors generally indicates that the disk drive is defective and requires repair.

System Crash Procedures

This section describes what a "crash" is, the types of crashes, and the appropriate measures to be taken.

A **crash** means that the UNIX operating system has suddenly ceased to function. Crashes may be associated with a **panic:** error message and/or the sudden loss of system-to-user interaction. Basically, there are two types of crashes: Recoverable and Unrecoverable. The common denominator between these two is that file

system damage usually results. The severity of the damage becomes the governing factor when judging the type of crash.

Crash Procedure

When a **panic:** message is associated with a crash, the system administrator should write down the entire message.

If the **panic:** message indicates a memory error has occurred, additional information such as the memory address in error is displayed. These types of errors generally indicate a hardware memory device failure, and repair involves replacing the defective device.

If the error condition exhibited is different from the above, then the system administrator can follow Figure 7-1, "Crash Procedure Flow Chart" for the proper sequence of actions.

Restoring the System after a Crash

This section outlines the steps for restoring the system to an operational state.

The utilities involved are those which were described in detail in Chapters 4 and 5; therefore, only the request to perform a specific program is given.

If a file system other than "root" has been corrupted, proceed to STEP 6; otherwise, go to STEP 1.

PROCEDURE: **How to restore the system after a crash**

1. Press the reset button.
2. Ensure the Onyx INIT tape is in the SAFE position, then insert it into the system.
3. Enter c (boot from tape) at the boot device query, then proceed to the standalone (**SHELL$$**) environment. Enter **init** to execute commands from tape.
4. Save a new standalone shell and load in the new root file system from tape, **but do not format the drive!**
5. Boot up the system to single-user mode and perform an **fsck** file system check for errors.
6. If another file system was damaged, perform the following; otherwise, GO TO STEP 7.
   (a) From single-user, execute the **mkfs(1M)** command to rebuild the file system.
   (b) Label the file system using the **labelit(1)** command.
   (c) Unmount the file system using the **umount(1M)** command.
   (d) Ensuring the tape is in the SAFE position, insert the most recent backup tape of this file system.
   (e) Check the file system using **fsck(1M)**.
   (f) Mount the file system using the **mount(1M)** command.
   (g) Restore the file system data from the tape using **cpio**.
   (h) If errors occur, repeat steps "a" through "g;" otherwise, go to STEP 7.
7. Resume normal operation.

Emergency Shutdown

This section describes how to terminate operations quickly on the system.

The specific methods discussed are: **shutdown**, **killall**, and "last resort."

Note: to bring down the system, the system administrator should either be logged in as root or become super-user from the console terminal.

The **shutdown(1M)** program will methodically, and in an organized manner, terminate all the processes currently running on the system. Shutdown notifies and affords all currently active users 60 seconds to prepare for this event. However, in an emergency the time factor can be defeated. Because of its attributes, the **shutdown(1M)** program is the prescribed primary method for terminating system operation.

PROCEDURE: **How to use the shutdown program**

1. Do the following:

   # /etc/shutdown 0 <cr>

   [Displays the various messages.]

   INIT: SINGLE USER MODE

   Current date:[date]
   #

2. Update the super-block, then press the reset button or remove power.

   # sync;sync <cr>
   #

Note: the "0" after the command defeats the 60 second delay. If not desired, simply do not enter the "0."

The **killall** program terminates all currently active users' processes. However, it does not notify users of the coming event, and it silently allows twenty seconds to end any work.

PROCEDURE: **How to use the killall program**

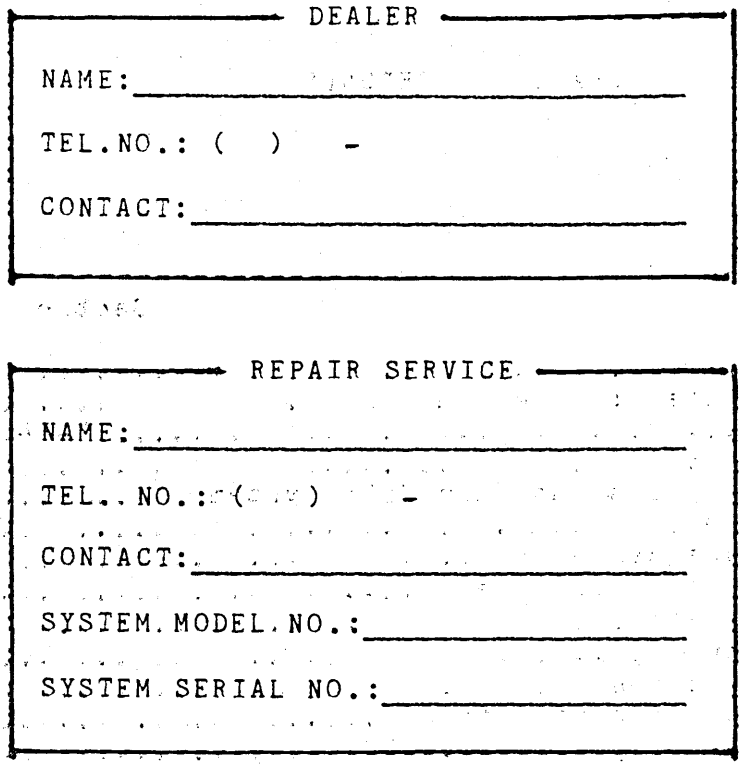
1. Do the following:

   # /etc/killall <cr>

   killed: [UID's]
   #

   # sync;sync <cr>
   #

2. The system is now ready for reset or power down.


DEALER
NAME:
TEL.NO.: ( ) -
CONTACT:
REPAIR SERVICE
NAME:
TEL. NO.: ( ) -
CONTACT:
SYSTEM MODEL NO.:
SYSTEM SERIAL NO.:

INDEX OF PROCEDURES