

Effluent Plant

CHARLES RIVER DATA SYSTEMS

**UNOS
System Concepts
Manual**

3890-778

UNOS System Concepts Manual

CRDS: 97-31992-02

October 1986

Charles River Data Systems
983 Concord St.
Framingham, MA 01701

All Rights Reserved. No part of this document may be reproduced in any form or by any means, electronic or mechanical, including photocopying or recording on electronic media without written permission.

This manual was written by Sandra Brown-Macioci.

Copyright(c) 1986 Charles River Data Systems

UNOS is a registered trademark of Charles River Data Systems

CRDS: 97-31992 Revision 02

Introduction

The *UNOS System Concepts Manual* explains software information. To use this information, you must install your system and boot the UNOS operating system. See your *Universe System Installation Manual* for these instructions.

The *UNOS System Concepts Manual* helps you, the system administrator, maintain the UNOS operating system and helps you complete routine maintenance tasks.

This manual explains procedures for configuring terminals and devices, creating user accounts, creating file systems, managing your system, and backing up and restoring files. This manual is set up to follow your progress as you set up and run your system.

Manual Structure

Follow the concepts manual chronologically.

Introduction

gives you an overview of the *UNOS System Concepts Manual*. The *Introduction* includes the manual's purpose, a description of the manual's contents, an explanation of how fonts are used, and an explanation of environments. The *Table of Contents* follows the *Introduction*.

Section 1 System Set Up

instructs you on how to attach and configure disks, tapes, floppies, terminals, printers, modems, and user accounts. *Section 1* has three chapters:

- *Device Organization*
 - includes four sections
 - * Device Nodes
 - * Disk Organization
 - * System Customization
 - * Removable Devices
- *Terminal, Printer, and Modem Connections*
- *User Accounts*

Section 2 System Administration

explains the structure of file systems, how file systems work, setting up a configuration file, process priorities, changing system priority defaults, and back up procedures.

Section 2 includes four chapters:

- *UNOS File System*
- *Configuration Files*
- *UNOS Priorities*
- *Backup and Restore Procedure*

Section 3 Reference

includes a glossary, a table of device numbers, and an index.

Typesetting Conventions

Throughout our manuals, Charles River employs the following typesetting conventions:

`<cr>` indicates a carriage return (the RETURN key).

`^X` indicates a control character (the CTRL key plus a character) typed in a single keystroke.

`$X` indicates the escape sequence (the ESC key followed by a letter) typed sequentially.

`^[X` indicates an ALT command

`@>` indicates the system prompt.

`ADMIN` indicates the superuser prompt.

Your prompt may differ from the ones shown. When you see the prompt indicator in an example, don't type it. We show it to give you a point of reference.

`[ ]` indicates an option. Anything enclosed in the brackets is optional.

indicates a number, for example, 5, 10, 128.

bold typewriter type indicates system messages, command names, and system prompts.

regular typewriter type indicates user input.

All UNOS command lines end with a carriage return.

Throughout this manual, we have put the first use of a new term in *italics*. This indicates that the term is defined in the glossary. Although, titles of manuals are also in italics, but they do not appear in the glossary.

The UNOS default shell is the Korn Shell. UNOS also provides a Bourne Shell, C Shell, and Command Shell.

Contents

iii Introduction

3 Section 1: System Set Up

5 Chapter 1: Device Organization

5 Device Nodes

6 Device Designation

8 Disk Organization

16 Disk Configuration

27 System Customization

34 Removable Devices

47 Chapter 2: Terminal, Printer, and Modem Connections

47 Connecting to the Software

50 Installing Printers on Serial Ports

55 Using Modems

61 Chapter 3: User Accounts

61 System Password File

63 Groups

65 Users

69 Section 2: System Administration

71 Chapter 1: UNOS File System

71 Structure of UNOS File System

84 Multiple File Systems

89 Chapter 2: The Configuration File

89 Configuration Parameters

98 Configuration Files Customization

105 Chapter 3: UNOS Priorities

106 The Way Priority Works on UNOS

107 Priority Commands and Subroutines

109 Tuning Priorities

111 Chapter 4: Backup and Restore

111 Backup

122 Restore

124 Examples

Section 3: Reference

131 Glossary

139 Device Node Table

147 Index

Tables

7 Major Device Numbers

28 CC-1 VCC-1 Major/Minor Numbering

29 Interphase V/SMD 3200 Controller/Formatter Numbering

29 Xylogics Numbering

31 VIOP Major/Minor Device Numbers

40 Tape Driver Device Nodes -- Halversa

44 Half-Inch Tape Device Table -- NCR

48 Terminals in the Termcap File

Figures

35 Naming Conventions for Floppy Diskettes

39 Unlabelled Tape

43 Half-Inch Tape Device Names -- NCR

72 File System Layout

74 Small Files

75 Large Files

76 Huge Files

77 Gargantuan Files

83 Directory Data Block

Section 1

System Set Up



Contents

5	Chapter 1: Device Organization
5	Device Nodes
6	Device Designation
6	Major Device Number
7	Minor Device Numbers
8	Disk Organization
11	Pointer to the Bad Block Map
12	Pointer to the Bootstrap
14	Standard Disk Information
14	Controller Specific Disk Information
15	Partition Map
16	Disk Configuration
16	How to Make User Partition
17	Disk Repartitioning
19	Partition Changing
27	System Customization
27	Controller Information
32	New Device Installation
34	Removable Devices
34	Floppy Formats
37	Tape Formats
47	Chapter 2: Terminal, Printer, and Modem Connections
47	Connecting to the Software
50	Installing Printers on Serial Ports
51	Change Baud Rate
51	Activate login Program
51	Identify Device Nodes with Specific Printers
53	Adjust Software for Your Printer
54	Create a Line Printer Daemon Process
54	Activate Printer Functions in Multi-User Mode
55	Using Modems
55	Modem Types
56	Modem Set Up
57	Dialing In
58	Using an Onsite Network
58	Signing Off
59	Working Hints

61	Chapter 3: User Accounts
61	System Password File
61	User's Name
62	Encrypted Password
62	Group
62	Network Machine
63	User's Home Directory
63	Command Interpreter
63	Groups
65	Users
65	Adding a New User
65	Deleting a User

Tables

7	Major Device Numbers
28	CC-1 VCC-1 Major/Minor Numbering
29	Interphase V/SMD 3200 Controller/Formatter Numbering
29	Xylogics Numbering
31	VIOP Major/Minor Device Numbers
40	Tape Driver Device Nodes -- Halversa
44	Half-Inch Tape Device Table -- NCR
48	Terminals in the Termcap File

Figures

35	Naming Conventions for Floppy Diskettes
39	Unlabelled Tape
43	Half-Inch Tape Device Names -- NCR

Chapter 1

Device Organization

This chapter explains various device organizations. These organizations include

- disk format
- floppy format
- tape format

The *Device Organization* chapter defines *device nodes* and how a device is designated. Device designation describes the major and minor device numbering.

Besides definitions and explanations, this chapter provides instructions on how to

- reconfigure your disk
- write a PVH
- write a bad block map
- back up your disk
- create a new device

Device Nodes

Device nodes are the way the system associates *physical devices* including magnetic media -- disks, floppies, and tape cartridges -- with the operating system. UNOS knows about different *peripheral devices* on the system. Every different device has a *driver* which is responsible for the communication between the device and the system. All physical disks and tapes are partitioned into logical devices -- UNOS only recognizes these *logical devices*. Each logical device can support one file system. A floppy disk is a logical device that can support one file system.

Drivers are linked into the *operating system* when the system is created. Device nodes connect UNOS file names to the corresponding Major/minor device number. Although the UNOS distribution system

contains all the device nodes necessary to run the system, users can create additional device nodes using the **makedev** command. For more information, refer to the *Adding Devices* section in this chapter.

Usually device nodes reside in the directory, **/dev**, although they can be anywhere in the file system. Commands using device nodes do not assume that they are located in **/dev**, but instead require the complete pathname.

Device Designation

Each logical device has a unique system id number. This id is composed of two numbers: the *major device number* and the *minor device number*. The major device number designates a particular driver, while the minor device number is used by the driver to determine which logical device it is addressing. Many logical devices can exist on a physical disk and each of these devices can contain a *file system*.

Major Device Number

The major device number is an index into a system table called the *device switch table*. The table has an entry for every driver in the system and the entries are numbered from zero up to the number of entries minus one. The default number of entries is 32, but you can add devices up to 64 entries in the device switch table.

The drivers listed in the table below have corresponding major device numbers. These initial device numbers display below. See the controller tables in *Adding Devices* for each controller's additional major device numbers, if any. Major device numbers are multiples of 64 from the initial number. For example, the first SASI ID of the first CC-1 controller has a major device number of 4; the second slot has the major device number of 68.

Major Device Numbering for Device Drivers

Device Driver	Major Device#
System Internals (/dev/null, /dev/kmem and /dev/tty)	0
tty's controlled by the main CPU or integral IOP	1
VIOP #0-#3	2
VIOP #4-#7	3
Winchester disks, floppies, tapes on CC-1 or VCC-1	4
Additional CC-1 or VCC-1 cards	5-7
Half-inch Tape Master tape drives	8
Xylogic model 450 SMD+ disk controller	13
Interphase 3200	14
None	15
User drivers	16-31

Minor Device Numbers

The minor device number passes directly to the particular driver for its own interpretation. In the case of Winchester disks and 1/4" tape, the minor device number refers to a physical partition. Both devices can have up to 31 partitions.

For example, the minor device number 0 refers to the first partition on the device. For floppies and 1/2" tape, the minor device number refers to a mode of use, for example,

- single versus double density for floppies
- normal versus streaming operations for tape

For a detailed explanation of usage, refer to the respective floppy or tape section in this chapter.

Disk Organization

A partition map on the disk describes 31 regions on the disk accessible through different minor numbers. Partitions or minor device numbers facilitate file system checking and backing up, and, in the case of the floppy-sized pieces (called *floppy images*), speed I/O access. At installation, unless you specify otherwise, the installation program creates default partitions on your disk.

All fixed disks have a volume header block or a physical volume header, called a PVH, that is written on block zero of the device. Charles River supplies tools to read and write the PVH information on the head of a disk. These tools allow you to change parts of the disk configuration to suit the needs of a particular disk, controller, and system. The PVH contains

- a pointer to the bootstrap
- a pointer to the bad block map
- standard information about the disk format
- controller specific information about the disk format
- controller parameters

The space set aside for the PVH, part map, bad block map, and *unboot* is reserved by setting the start of partition 0 at some point after block 0, thus removing the first few blocks of the disk from any partition. This procedure gives a space with adjustable size as needed for each disk type, or even for an individual disk, without changing the drivers or the other programs that deal with PVHs.

A default size of 12 blocks is reserved for the PVH space. The 12 blocks provide enough space for all currently supported disk types. Having this space system adds some protection against accidental erasure or modification of this information because it is not accessible through the normal disk partitions.

To access the area of the disk outside the partition map all the current UNOS disk drivers recognize partition number 31 as a special partition which maps the entire disk, regardless of the partition

map. [1] Now, the PVH, partition map, bad block map, and bootstrap are accessible as the first few blocks of this partition.

Partition 31 is the only valid partition to use with the commands

- `copy_boot`
- `fmtdisk`
- `print_bad_blocks`
- `print_partition`
- `print_pvh`
- `write_bad_blocks`
- `write_partition`
- `write_pvh`

and the FORMAT `spfun()`. Also, if the disk has no partition map, partition 31 is the only valid partition for that disk. The conventional name for this special partition is `/dev/r[xx]all`. `xx` is the usual two letter type code for the disk. For example,
`/dev/rcqall` 32-MByte
`/dev/rcmall` Maxtor 118-MByte

unboot, the current version of the bootstrap routine, recognizes the partition map in order to find the start of the partition being booted. Consequently *unboot* can boot from any partition, not just partition 0. The new bootstrap routine boots all currently supported devices.

1. Partition 31 had previously been an invalid partition number.

A *pvh_file* tells the system the disk type and the disk configuration. For example, the *pvh_file* for a 130-MByte Winchester looks this way:

```
"130M_Fujitsu"  
"/dev/rckall"  
cylinders:823  
heads:10  
sectors:58  
bytes:256  
reduce:128  
stepperperiod:50  
stepwidth:3  
stepmode:0  
controller:SC3011  
interleave:2  
recovery:0  
option1:0  
option2:0  
option3:0
```

To display your disk's PVH, use this format

```
/sys/print_pvh device
```

For example, to display the *pvh_file* of a 32-MByte Winchester type

```
ADMIN>/sys/print_pvh /dev/rckall
```

NOTE: Neither **print_pvh** nor **write_pvh** applies to floppy disks or tape.

To install a *pvh_file*, you must be ADMIN. Use the command format

```
/sys/write_pvh pvh_file device
```

If the device already has a *pvh_file*, the **write_pvh** command asks you if you want to overwrite it. If you answer **Y** or **Yes**, **write_pvh** overwrites the original *pvh_file*. Use the RAW device as the device argument with the **write_pvh** command. For example, to install a physical volume header on a 32-MByte Winchester disk, type the command:

```
ADMIN>/sys/write_pvh /sys/pvh/cd /dev/rcdall
```

See the `write_pvh` page in the *UNOS Commands Manual* for more information.

Pointer to the Bad Block Map

The `pvh_file` contains the disk address and length of the *bad block map*. A bad block is an unusable area of the disk. You must identify and name the bad blocks in a bad block map. Usually, the manufacturer of the disk sends a bad block list with the disk.

Your system reads the bad block maps and avoids putting data in those blocks. The format to print your bad block map is

```
/sys/print_bad_blocks dev_path1 . . . dev_pathn
```

For example, to see the bad block map for a 40-MByte Quantum disk drive, type the command line:

```
@>print_bad_blocks /dev/rcqall
```

The system responds with the display:

```
" Bad block list for 40M Quantum disk drive.
" 3 defects * 17 blocks/defect = 51 unusable blocks.
" 69581 remaining usable blocks.
" Cylinder #      Head #
           28         0
           76         0
          125         1
```

Use a RAW device for the device argument. See the `print_bad_blocks` page in the *UNOS Commands Manual*.

The bad block map can be any size up to 2000 blocks, which is important when using large capacity disks that can have a large number of bad blocks. The pointer allows the operating system and bootstraps, which pay attention to bad blocks, to find the bad block map.

The **write_bad_blocks** command writes the bad block information to the disk and then identifies the pointers in the PVH. To create or overwrite a bad block map, you can create a file containing two kinds of lines:

1. A line containing two numbers separated by spaces or tabs represents the cylinder number and track (head) number of a known defect. The numbers are decimal, unless they start with 0 (octal) or 0x (hexadecimal).
2. A line beginning with the double quote character (") is a comment line. **write_bad_blocks** prints the line (on standard output), but otherwise ignores it.

You can only write a bad block map to disks with valid *pvh_files*. For example, the file *bblist* contains the bad block areas for a 40-MByte Quantum disk.

"Bad block list for 40Meg. Quantum disk drive.

28	0
76	1
125	1

To write the bad block map to the disk, type this command line:

```
e>write_bad_blocks bblist /dev/rcqall
```

The system responds with

"Bad block list for 40Meg. Quantum disk drive.

Bad block record (3 items) written on

40m_Quantum_Winchester /dev/rcqall

NOTE: Do not alter the bad block map on disks that have file systems because the files and directories get scrambled.

See the **write_bad_blocks** page of the *UNOS Commands Manual* for more information.

Pointer to the Bootstrap

The *pvh_file* contains the disk address and length of *unboot* and a *magic word*, currently *EXORMACS*, that validates this disk as a bootable disk. *Unbug* uses this information to find and load the

bootstrap when you issue the **boot** command.

The **copy_boot** command does the following to the *pvh_file*

- does a check for overwriting a filesystem header. If you are about to write over the file system, you receive a warning
- writes the unboot bootstrap to the disk immediately after the bad block map
- writes the pointers and the magic word, *EXORMACS*, at the head of the block

copy_boot does not update any other information in the *pvh_file*.

The unboot program works with all three types of disk controllers:

- CC-1/VCC-1
- Xylogics/Eagle
- Interphase/NEC

and all types of CPUs.

unboot boots from a user specified partition that allows the bad blocks map, the PVH, and the bootstrap program to be in a special invisible area. The unboot program reads the partition map if one is provided to find out where the bootstrap partition is.

UNBUG allows *unboot* to boot partitions other than 0 on a specified device. *unboot* understands

- the PVH format that placed bad blocks in block 2
- the PVH format that points to the bad blocks
- the one block bad block format
- the multi-block bad block format

unboot contains drivers for Xylogics and CC-1 devices internally which are only used with Release 3.0 or earlier. *unboot* uses any UNBUG provided driver that conforms to the current interface standard.

Standard Disk Information

The *pvh_file* contains standard disk format information. Standard information includes

- the number of tracks per cylinder
- the number of cylinders
- the number of sectors per track
- the interleave factor
- the interleave recovery time

interleave factor is the number of sectors that pass between two consecutive read or write commands to allow the disk controller firmware time to setup for a new command.

Controller Specific Disk Information

In addition to the standard disk information, the PVH contains controller specific information required to initialize a disk controller to properly handle the disk. This information is in a standardized format that pertains mostly to SCSI controllers. Because future controllers may require more or different information, three option fields are provided for this purpose.

The controller specific information includes a 22-byte ASCII drive name, and an 8-byte ASCII controller name.

The PVH also includes

- *stepper pulse* width
- stepper pulse period
- stepper pulse mode
- maximum head
- reduced write current cylinder
- a drive ID byte (known to controller)
- a *device name*

The information allows the device initialization routine in a driver to set up the drive parameters in a standard manner and read the PVH (that only requires that one can read block 0). Once the PVH has been read, the controller can be reinitialized with the correct information required to read the rest of the disk.

The operating system uses this information to initialize disks, both in the kernel and the Unbug disk drivers used by *unboot*. This initialization process allows you to attach a variety of disk drive types to the system simply by updating the PVH. Adding new disk controllers that require different software, because of variations in their command formats, still require modification to the drivers, the kernel, and UNBUG. Use the **write__pvh** command to write the controller specific information.

Partition Map

The partition map describes the base and bounds of each partition located on a device.

The partition map is written by the **write__partition** command and its contents can be read by the **print__partition** command.

Default partitioning differs from system to system. See *Appendix B* of your *Universe System Installation Manual*. The default partitions listed below are the usual defaults.

first partition	reserved for the back-up root
second partition	reserved for swapping
third partition	reserved for the main root

You can use the other partitions as you want.

To change the default, see the *Partition Changing* section of this chapter. You can change the number of partitions, the partition's sizes and the partition's location on the disk.

Some older systems have subsections that are 1-Mbyte large. These floppy image areas are sections of the disk with the same number of blocks as an actual floppy. You can write to this area in a piecemeal fashion and then transfer the entire image to the floppy in a single operation. Time is saved, because there are no repeated I/O operations to the slower floppy disk.

Configuring the Disk--Not Applicable to removable media

Before you install any users, you should consider whether you want to mount the user section on a separate disk partition. We recommend separate partitions for several reasons:

- provides more room for user files
- makes the necessarily frequent user backing up easier
- avoids redundant back ups of the unchanging root

How to Make User Partition

Refer to the Major/minor device configuration table under the *Major Device Node* heading in this section of the manual and decide where you want to place the user partition.

Use the **makefs** command to create a UNOS file system on the specified device. At the beginning of the device, **makefs** writes the *file system header*, the *block bit map*, and the *file descriptors*. All file system layouts are the same, except that the amount of space allocated for the bit map, the file descriptors and the user space is a function of the size of the media. (Refer to the *UNOS File System* chapter for more about **makefs**.)

In the following example, you make a file system of 20,000 blocks on device **/dev/cq3**.

```
ADMIN>makefs /dev/cq3 20000
```

Edit the file **/etc/startup**. This script file is run when you switch from single to multiuser. It starts the update program, cleans up temporary files, mounts devices, fixes printer queues and starts the printer daemon. Insert a line that mounts the user section of the disk. (In the following file, we have indicated the added line with an arrow, ->, do not type this arrow.)

```
echo entering startup file
/sys/update &
delete -force -silent /tmp/[_!]*
echo leaving startup file
->mount /dev/rck3 /usr
```

Disk Repartitioning

The *Disk Repartitioning* section explains why you would repartition, and how to repartition a disk. The first part of this section describes what to do before you change your disk partitioning. This involves making a back up of your current configuration. The next section describes the **write_partition** command. The last part, *Examples*, presents two problems that repartitioning can solve.

Step-by-step instructions accompany each solution.

Reasons for Changing Your Default Partition

Changing the default partitions on your disk can solve space problems and streamline file systems. Reasons for changing your default partitioning include

- running out of space on a partition
- unused space on a partition
- backing up stagnant files repeatedly

Before repartitioning your disk review the following commands:

package

backs up your disk to floppy disks or tape.

makedev

creates a new device node. Just creating a new partition in the partition map is not enough. You must use the **makedev** command to create a new device node.

makefs

creates a new file system. Whenever you change the size of a partition, you must remake the file system so the system knows it has that much space on a certain partition. Just creating a partition in the partition map is not enough.

write_partition

writes a partition map onto the disk. You can use the default partition map by using your disk partition file, or you can create your own partition map. Explanations for both choices appear in this section.

To reconfigure your disk, look first at the device table for your disk in *Appendix B* of this manual. These tables show the default disk partition maps written on the disk during installation.

Notes Before You Repartition

- Back up your entire disk using the **package** command.

This example uses the **package** command to back up a file system to floppy disks. Make sure you mount all the partitions you want to save. An unmounted file system is not saved. **package** backs up the file system from the root, /, and puts the file system on **dev/rcfd0**.

```
ADMIN>package / > /dev/rcfd0
```

To use the **package** command to back up to tape use the **-bs** option. The **-bs** option denotes a smaller buffer size than the default. Using a smaller buffer size can increase the usage of the tape. You can also use the **-bs** option with floppy disks.

```
ADMIN>package -bs=256 > /dev/rct0
```

or

```
ADMIN>package -bs=256 > /dev/rct0
```

Use the **installation** command to restore the information. See the *UNOS Commands Manual* for more information.

- Know the size of your disk
- Create no more than 31 partitions
- Start partition zero at block twelve and make it at least 6 blocks long

-
- Work in single user mode
 - Make sure the main root is `/dev/rc[a,b,c,d,j,k,m,q,s,t]2`
 - Make sure nothing is mounted

Partition Changing

You can change the partitions on your disk with the **write_partition** command. This command lets you slightly modify or completely change the disk partitioning.[2]

The **write_partition** command writes the partition table in the second half of block zero on the disk (in the IID area). [3]

The format for the **write_partition** command is
`/sys/write_partition [offset=block number] [partitions]
device`

The *offset=block number* adjusts the starting block of partition 0.
partitions is a file that includes a disk's

- minor device number
- starting location of the partition in blocks
- limit of the partition in blocks

You can specify the default partition table for your disk by using the the **write_partition** command with the *partitions* file. By choosing one of the disk names listed below, you receive a default partitioning for that disk. The *partition* file has *all* appended to the device name. The **device** name refers to the hard disk drive. For example, `/dev/rcqall` refers to the *partition* file for the Quantum 40-MByte disk.

2. A full explanation of the **write_partition** command appears in the *UNOS Commands Manual*.

3. Refer to the *UNOS File System* chapter in this manual for a description of the file system layout.

The **write_partition** command requires a RAW device as its device argument. For example:

ADMIN>/sys/write_partition partfile /dev/rcdall
works whereas,

ADMIN>/sys/write_partition partfile /dev/cdall
fails with this error:

/sys/write_partition: Device '/dev/cdall' is not a disk

The *partition* files comes from the following disk names:

ca NEC 20-MByte
cb Seagate 10-MByte
cc Quantum 16-MByte
cd Quantum 32-MByte
cj Fujitsu 60-MByte
ck Fujitsu 130-MByte
cm Maxtor 118-MByte
cq Quantum 40-MByte
in NEC 450-MByte
xe Fujitsu 350-MByte
xn NEC 400-MByte

Appendix B in this manual has the default partitioning for the disks.

For example, if you type

@>/sys/write_partition partfile /dev/rcqall

your partition map for the Quantum 40-MByte disk would look like this

"partition	offset	limit
0	12	2000
1	2000	4000
2	4000	24000
3	24000	44000
4	44000	64000
5	4000	44000
6	64000	68000

If you create your own partition file, it must contain all the partitioning for the disk.

The partition file should contain a line for each partition on the disk. Each line has three parts:

1. the partition number, with a maximum of 31 partitions, numbered 0-30
2. the starting location (offset)
3. the end of the partition (limit)

In the example above of the 40-MByte Quantum disk, every line of the partition map includes a partition number, a starting location, and the end of the partition.

- The first line names the partition number as 0, the starting location as 12, and the end of the partition as 2000.
- The second line names the partition number as 1, the starting location as 2000, and the end of the partition as 4000.

The offset and limit are in blocks. The minimum size of a partition is one block; therefore, if the starting location is equal to the size of the partition, then the partition is zero length and hence, not accessible. You can insert any comment lines by beginning the line with a double quote (").

If you increase the swap space, increase the **cnf_swapspace** parameter in the configuration file. Otherwise, the configuration file and the kernel don't know the swap space increased.

CAUTION

Make sure you add correctly, drop no zeros, and use overlapping partitions correctly. The **write_partition** command does not determine disk size, cannot check for overlapping areas, and does not know whether data areas have been allocated.

For more detailed information about the **write_partition** and **print_partition** commands, refer to the *UNOS Commands Manual*.

Examples

This example section presents two problems you can solve by repartitioning.

-
1. Running out of space on a partition
 2. Speeding the back-up procedure

Partition space problems require increasing the memory in one partition and decreasing the memory in another. Speeding the back-up procedure requires creating a new partition and taking memory from an existing partition for the newly created one.

The solutions to these problems include step-by-step explanations for repartitioning.

Space Problems Solved by Repartitioning

If you run out of space on a partition, you have a few options to remedy the problem.

- Delete files. Maybe you have a backlog of *.bak* files or files that are no longer current.
- Copy files to tape or floppy. Files you use infrequently can reside off the disk until you need them. After you copy the files, delete them from the disk partition.
- Repartition your disk

Assuming you choose to repartition your disk, you'll need to find a partition with more space than it needs. For this example, you have a 32-MByte Winchester, and you want to increase the swap partition, partition 1, by 3000 blocks. You determine that partition 4 needs only 2000 blocks rather than 5000 blocks.

NOTE: Be sure you meet all the requirements in the *Notes Before You Repartition* section before you begin to repartition your disk.

Follow these directions to repartition your disk as specified above.

1. Change to **ADMIN**.
2. Create a file, *partfile*, that describes the partition map.
Type
ADMIN/sys/print_partition /dev/rcdall > partfile

3. Edit the *partfile* with your editor. Type **ADMIN> ted**
partfile

"partition	offset	limit
0	12	2000
1	2000	4000
2	4000	30000
3	30000	34000
4	34000	39000
5	39000	44000
6	44000	54000
7	54000	56000
8	56000	58000
9	58000	64000

Change the partitions as shown below:

"partition	offset	limit
0	12	2000
1	2000	7000
2	7000	33000
3	33000	37000
4	37000	39000
5	39000	44000
6	44000	54000
7	54000	56000
8	56000	58000
9	58000	64000

Notice that partition 1 increased by 3000 blocks, and
partition 4 decreased by 3000 blocks.

Tell the disk to replace the old *partfile* with the new one you just
created. Type

ADMIN>/sys/write_partition partfile /dev/rxdall

The system responds

The disk already has a partition record.

Would you like to overwrite it?

```

Type
yes or y
"partition          offset          limit
      0              12             2000
      1             2000            7000
      2             7000           33000
      3            33000           37000
      4            37000           39000
      5            39000           44000
      6            44000           54000
      7            54000           56000
      8            56000           58000
      9            58000           64000

```

"Partition map written

4. Because you changed the size of the partition, you must also recreate a file system with file descriptors that match the new size of the partition. You can make a file system smaller than the partition its in, however, you cannot make a file system larger than its partition.

Create a file system for partition 1 with 5000 blocks.

```

Type
ADMIN>makefs /dev/rcd1 5000
Making filemap
Making fdmap with 715 fds
Making new fds
4089 blocks available writing headers

```

Now, create a file system for partition 4 with only 2000 blocks.

```

Type
ADMIN>makefs /dev/rcd4 2000
Making filemap
Making fdmap with 288 fds
Making new fds
1955 blocks available writing headers

```

Back-Up Procedure Made Easier

Look at your `/usr` directory. Directories, like `/usr/lib` and `/usr/bin`, change only with releases. Rather than back up `/usr/lib` and `usr/bin` everytime you back up the `/usr` partition, repartition the disk so `/usr/lib` and `/usr/bin` have their own partitions.

NOTE: Be sure you meet all the requirements in the *Notes Before You Repartition* section before you begin to repartition your disk.

To create a new partition, partition 9, for stagnant files on a 130-MByte Fujitsu, follow these directions.

1. Change to **ADMIN**.
2. Create a file, *partfile*, that describes the partition map.
Type
ADMIN/sys/print_partition /dev/rckall > partfile

Edit the *partfile* with your editor.

Type

ADMIN>ted partfile

"partition	offset	limit
0	12	2000
1	2000	10000
2	10000	42000
3	42000	114000
4	114000	186000
5	186000	206000
6	206000	208000
7	208000	218000
8	218000	228000
9	0	238670

Change the partitions as shown below:

"partition	offset	limit
0	12	2000
1	2000	10000
2	10000	42000
3	42000	114000
4	114000	186000
5	186000	206000
6	206000	208000
7	208000	218000
8	218000	228000
9	228000	234000
10	0	238670

Notice the addition of partition 10. Partition 9 has changed from the whole disk partition to a 6000 block partition. Partition 10 is now the whole disk partition.

Tell the disk to replace the old *partfile* with the new one you just created. Type

ADMIN>/sys/write_partition partfile /dev/rckall

The system responds

The disk already has a partition record.

Would you like to overwrite it?

Type

yes or y

"partition	offset	limit
0	12	2000
1	2000	10000
2	10000	42000
3	42000	114000
4	114000	186000
5	186000	206000
6	206000	208000
7	208000	218000
8	218000	228000
9	228000	234000
10	0	238670

"Partition map written

3. Create an unbuffered device using the major device number 4, for partition 10. Because partition 10 did not exist previous to repartitioning, you must create it with the **makedev** command. Type
ADMIN>makedev /dev/rck10 u 4 10

Because the file system on partition 9 changes, you need to make a new file system on that partition. To change the file system on partition 9, use the **makefs** command. Type

```
ADMIN>makefs /dev/rck9 6000  
making freemap  
making fdmap with 568 fds  
making new fd's  
3920 blocks available, writing header
```

CAUTION: Do not use the **makefs** command on the device name that means the whole disk or you will delete the first file system on the disk.

Put the stagnant files in **/dev/rck9** and back up **/dev/rck9**. You have more room on your **/usr** partition.

To back up your partitions, follow the directions in *Notes Before You Repartition*

System Customization

UNOS allows you to add more logical and physical devices to your system, and also allows you to change the default partitioning on your existing Winchester disks. This section discusses adding new disks and changing your disk partitioning.

Controller Information

UNOS supports four CC-1 or VCC-1 cards per system. Each CC-1 or VCC-1 card supports four controllers, which in turn support up to four devices. Two bits are borrowed from the major number, to permit 31 partitions on each of 4 disks on each of 4 controllers on a given CC-1 or VCC-1 card (4 cards supported). The CC-1 VCC-1 Controller Table indicates the major and minor device numbering.

When you add controllers and devices, your Major/minor numbers change. Refer to the following table for an overview of the numbering scheme. The Major/minor device numbers for the VCC-1 card are the same as for the CC-1 card.

CC-1 VCC-1 Major/Minor Numbering

[V]CC-1	Controller	Devices			
	Major #	Minor #			
		dev0	dev1	dev2	dev3
0	4	0-31	32-63	64-95	96-127
	68	0-31	32-63	64-95	96-127
	132	0-31	32-63	64-95	96-127
	196	0-31	32-63	64-95	96-127
1	5	0-31	32-63	64-95	96-127
	69	0-31	32-63	64-95	96-127
	133	0-31	32-63	64-95	96-127
	197	0-31	32-63	64-95	96-127
2	6	0-31	32-63	64-95	96-127
	70	0-31	32-63	64-95	96-127
	134	0-31	32-63	64-95	96-127
	198	0-31	32-63	64-95	96-127
3	7	0-31	32-63	64-95	96-127
	71	0-31	32-63	64-95	96-127
	135	0-31	32-63	64-95	96-127
	199	0-31	32-63	64-95	96-127

UNOS can support a maximum of four Interphase V/SMD 3200 Controller/Formatters with each controller supporting two Winchester disks on VMEbus systems. The Interphase V/SMD 3200 Controller/Formatter Numbering table describes the Major/minor device numbering for the controller.

Interphase V/SMD 3200 Controller/Formatter Numbering

Controller #	Major Device #	minor device #	minor device #
0	14	0-31	32-63
1	78	0-31	32-63
2	142	0-31	32-63
3	206	0-31	32-63

For VERSAbus systems, UNOS supports one Xylogics driver. The Xylogics driver can support four Winchester disks. The Xylogics Numbering table describes the Major/Minor device numbering the driver.

Xylogics Numbering

Major Device	minor device #
13	0-31
13	32-63
13	64-95
13	96-127

Charles River's VMEbus systems can accept additional terminals through additional Input/Output Processors (VIOPs). The terminal devices, `/dev/tty0` through `/dev/tty3`, use the SCC driver. The terminal devices reserve major device #1 for SCC devices.

Any additional tty devices are attached to the optional VME VIOP, which uses the mbx driver software. Since major device #1 is already reserved, the mbx driver recognizes major device #2 for its tty devices. Any tty devices attached to the first optional VIOP use major device #2. Each VIOP can support up to 64 terminals.

tty devices. Any tty devices attached to the first optional VIOP use major device #2. Each VIOP can support up to 64 terminals.

A VCP2000 system can have up to eight VIOPs installed. The following table shows the the major device numbers you must use for tty devices you attach to the first four optional VIOPs installed on one VCP2000. You must know the major device number for the **makedev** command.

VIOP Major/Minor Device Numbers

Binary								Decimal	
VIOP#				Major Device#				VIOP#	Major Device#
0	0	0	0	0	0	1	0	0	2
0	1	0	0	0	0	1	0	1	66
1	0	0	0	0	0	1	0	2	130
1	1	0	0	0	0	1	0	3	194
0	0	0	0	0	0	1	1	4	3
0	1	0	0	0	0	1	1	5	67
1	0	0	0	0	0	1	1	6	131
1	1	0	0	0	0	1	1	7	195

The arrow shows that the least significant bit of the major device number becomes the most significant bit of the VIOP number.

The major device number structure allows the mbx driver to determine which board a particular tty device uses. The binary major device number tells the mbx driver to use major number 2 in the lower six bits. The least significant bit of the major device

number becomes the most significant bit of the VIOP. The VIOP board number is encoded in the high two bits. The actual major device number (whose decimal number you must plug in to the **makedev** command when you create tty devices) is the combination of the two.

You can name VIOPs as follows:
`/dev/ttyMMMM`

The major device number has three characters and the minor device number has two characters. Minor device numbers 0-3 represent ports on the VIOP board. Minor device number 0 represents the first port on the board. Minor device number 3 represents the last port on the board. MUX board designation begins with minor device number 4. For more information on MUX boards, see the *VIOP-1-2 User's Manual* and the *VME Serial Multiplexer Board User's Manual*.

Examples of naming VIOPs are
`/dev/tty00200`

002 indicates the first VIOP board. 00 indicates the first port on the board.

`/dev/tty19403`

194 is the fourth VIOP board. 03 indicates the last port on the board.

The VIOP Major Device Number Figure illustrates how to determine the numbers.

After you have decided how many controllers and how many devices you'll be adding, and have determined the correct Major/minor numbers, you have to associate the device with the system.

New Device Installation

Before you can add a device to the operating system, you must first have a device driver. In some cases, such as adding another tty, you can use the existing driver. In other cases, however, you must first write a driver. UNOS provides the necessary drivers for general operation; only if you want to add a different peripheral do you

need to write your own driver. For more information about writing drivers, refer to the *UNOS Guide to Device Drivers*.

makedev

All drivers require corresponding device nodes. The **makedev** command creates a device node. Device nodes reside in the **/dev** directory by convention.

The following is a typical entry in the **/dev** directory on a 68/05:

```
B 4, 0 rw.  rwx rwx (1) admin Apr 25 16:31:09 cb0
```

The fields have the following meanings:

1. The first field indicates whether to access the device in *buffered* (B) or *unbuffered* (U) mode. You can access many devices in either buffered or unbuffered mode, but you can only access tty ports in *unbuffered* mode.
2. The second field refers to the device driver's type. This field is the device's major device number.
3. The third field refers to the device's minor number.
4. The following fields refer to the permission modes, number of links, owner, creation date, and name of the device.

The **makedev** command has this syntax:

makedev name [u or b] major minor

The command requires a full pathname for a device name, either an *unbuffered* or a *buffered* specification and the Major/minor device numbers.

Refer to *Appendix B* for a listing of the UNOS Major/minor device designations.

An entry for a 130-MByte Fujitsu would look like this:

```
@>makedev /dev/rck10 u 4 10
```

Removable Devices

Removable devices include floppy disks and tape. Usually, these media are used as back up devices and not interactive ones. Charles River has 8" and 5 1/4" floppy disks and 1/4" and 1/2" tape. The *Floppy Format* section and the *Tape Format* section give explanations of these devices.

Floppy Formats

The floppy disk node you specify determines how UNOS writes data onto the disk. The floppy disk has several different formats:

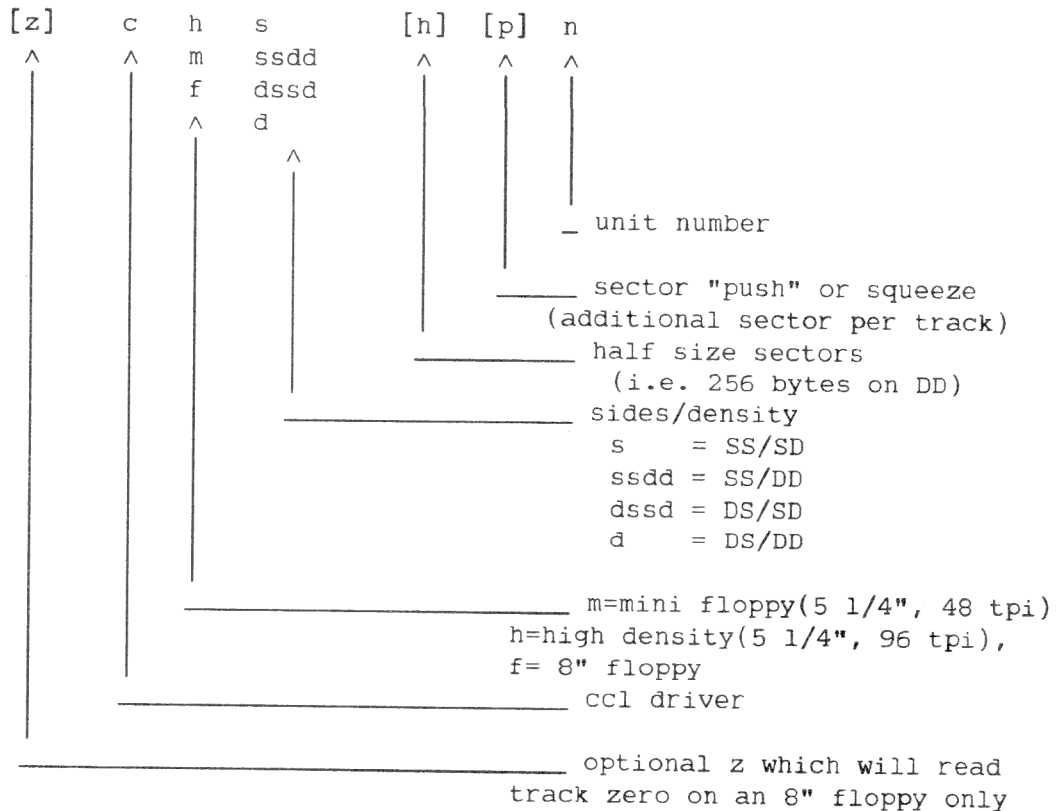
- Zero- or non-zero based--controls whether the first track (track 0) is used
- Single- or double-sided
- Single- or double-density

Naming Conventions for Floppy Diskettes

The floppy devices are named in the following manner:

The generic device name is **c** (for CC-1). Zero-based floppies are always single-density.

Naming Conventions for Floppy Diskettes



For example, the `/dev/cmssddhp0` means

- CC1 driver
- mini-floppy
- single-sided
- double-density
- half-size sectors (256 bytes on dd)

-
- sector push
 - unit number

Because the IBM standard reserves track zero on 8" floppies for volume information, UNOS normally skips track zero when using 8" floppies. If you need to write to, or read from track zero, choose the corresponding floppy device that begins with the letter **z**, for example, `zcfs0` or `zcfdsd0`.

NOTE: `/dev/cfd0` is the only 8" floppy drive accessed as a double-sided/double-density disk and skipping track zero on the first side.

The two device types for 5 1/4" floppy are

1. 5 1/4", 96 tpi[4] also called 5 1/4", 1.2 Mbyte
2. 5 1/4", 48 tpi also called 5 1/4", 320 Kbyte

Specify the drive type and format with the minor device number used to access the drive. If you want a 96 tpi drive to have 48 tpi drive compatibility, use the minor device numbers for tpi compatibility listed in *Appendix B* of this manual.

The default names used for 48 tpi compatibility mode correspond with the default 48 tpi names. No naming conflict results, because no system has both the 96 tpi drives and the 48 tpi drives. A complete list of drive types and formats with their corresponding minor device numbers and conventional names appear in *Appendix B* of this manual.

The 48 tpi compatibility mode formats as read only. The software for 48 tpi compatibility mode will allow writing and formatting of disks in compatibility mode. However, Charles River does not guarantee that such disks will be readable on any other system.

4. *tpi* means track per inch

Tape Formats

Charles River supports 1/4" tape cartridge tape drive and two 1/2" tape drives.

Quarter-Inch Tape

You can read a quarter-inch tape cartridge tape drive just like a disk. In other words, any block can be read from any of the files recorded on the tape, although such random access is extremely slow.

The minor number in the device node (0-15) is the number of tape marks on the tape to skip over in order to read the specific file. UNOS provides nodes for the first eight partitions on a tape, **/dev/rct0-7**. Of course, the actual size of each partition is determined when the partition is written.

When you write to the quarter-inch cartridge tape, you can only write to the logical beginning of the tape which is at the physical beginning of the tape or after a valid tapemark. For example, the tapemark at the end of **ct0** is the logical beginning of the tape for **ct1**. The last record written is always the logical end of the tape. For example, the **debe** command to **/dev/rct1** skips over the first partition on the tape (which must have been previously written), overwrites all of the second tape partition, and makes all subsequent partitions on the tape inaccessible.

UNOS stores tape data in 512-byte blocks regardless of the length of any single write request.

To bypass this tape positioning, access the tape via **/dev/rawct0**, using the minor device number 144. This action disables the tape positioning.

Half-Inch Tape

Charles River's half-inch magnetic tape can store up to 90 Mbytes on a 10 1/2 inch tape. The half-inch tape comes in three reel sizes, 7", 8 1/2" and 10 1/2" featuring a density of 1600 bpi and 3200 bpi.

A standard tape reel holds 2400 feet of tape, which will store about 45 Mbytes of data at 1600 bpi. However, a .6 inch gap between records reduces the capacity drastically for short records. For example, with 512-byte records, the tape stores only about 15 Mbytes. 90% tape utilization requires at least 8500 bytes per record at 1600 bpi and proportionately more at higher densities.

Charles River supports two types of half-inch magnetic tape:

- a Cypher F880 Microstream Tape Drive and supports a TapeMaster A/ HalVersa Synergist I Rev. E controller
- the NCR H6210 controller, which talks to the Cipher 1/2" tape drive uses the new SCSI-standard INQUIRY command to find out what kind of controller is at the other end of the SASI bus

Operating times and data movement information include:

- 39 Kbytes/sec. for tapes with 1600 bpi density and 25 ips operational speed
- 159 Kbytes/sec. for tapes with 1600 bpi density and 100 ips operational speed
- 159 Kbytes/sec. for 3200 bpi density and 50 ips operational speed

Half-inch tape can operate in three modes:

- buffered mode
- direct mode
- streaming operation

Changing tape density in the middle of a tape is unsupported, so you must write a particular tape in one density throughout.

Although you can write to the tape only in a forward direction, you can read tape in either forward or reverse direction. Half-inch tape includes a phase encoded format with 9 tracks, 8 for data and 1 for parity.

Half-inch tape can be formatted in two ways: labelled and unlabelled. You can read tape that is either labelled or unlabelled. However, Charles River does not create tapes with a labelled format.

The 1/2" tape drivers read either labelled or unlabelled tape provided the tape meets the density requirements. The UNOS half-inch

magnetic tape command, **tp**, reads the labels on the labelled tape as data files. If you use the **tp** command to write or copy onto a tape, the tape format is unlabelled.

Unlabelled Tape Format

Unlabelled tape format consists of file sections and tape marks. The file sections include the necessary information to identify the file on tape. Tape marks separate the file sections from each other. Double tape marks indicate the end of the volume. Use the UN/IX tools commands **cpio** and **tar**, and the UNOS commands **tp**, **debe**, **installation**, **iostats**, and **package** to work with unlabelled tape.

The default device for **tp** is

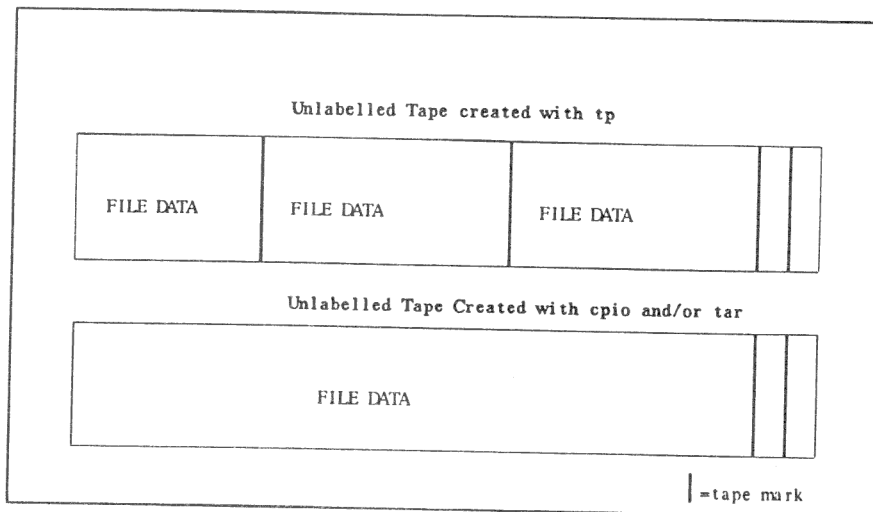
/dev/mt.async u 8 4

See the **tp** command page in the *UNOS Commands Manual* for options and special properties of the Tape Master and SCSI controllers.

The default device for **debe** and **cpio** is

/dev/mt0 u 8 0

Unlabelled Tape



cpio and **tar** are UNIX tools that expand the capabilities of your half-inch tape. If you create an unlabelled tape with **cpio** and **tar**, you must read and write using **cpio** and **tar**. These UNIX utilities create only one tape mark. The one tape mark indicates the end of the tape. [5]

This table shows the Major/minor device numbers for half-inch tape using the Halversa card, the device names, the purposes for those names and whether or not the tape is buffered.

Tape Driver Device Nodes - Halversa

MAJOR DEVICE	MINOR DEVICE	NAME	PURPOSE	u/b
8	0-3	/dev/mt0-3	normal operation	u/b
8	4-7	/dev/mt.async0-3	asynchronous tape interface	u/b
8	8-11	/dev/nrmt0-3	no rewind	u/b
8	16-19	/dev/stream0-3	streaming operation	u/b
8	32-35	/dev/nfmmt0-3	ignore filemarks	u/b
8	40-43	/dev/rawmt0-3	no rewind/ignore filemarks	u/b
8	84-87	/dev/conmt	long inter-record gaps	u/b
8	128-131	nemt0-3	no EOF written at end of tape	u/b

5. For further information about **cpio** and **tar**, see the *UNIX TOOLS RELEASE*.

The NCR controller half-inch tape must have its own CC-1, and be at SASI ID 0, so the major device number is 5 with the minor numbers starting at 128 because the high bit is on, meaning removable medium. An example of a half-inch tape Major/minor device table appears at the end of the chapter.

The driver limits you to 4 tape drives, each with its own controller, per CC-1 card, and to four CC-1 cards for a whole system. Since you need one CC-1 card for a disk, the limit is twelve tape drives per system.

UNIX requires the two new directories **/dev/mt**, for buffered tape device nodes, and **/dev/rmt**, for unbuffered (raw) tape device nodes.

Unbuffered devices transfer data directly to or from buffers provided by the user program, but the transfer can be rejected for various hardware-related reasons: buffer at odd address or buffer size not multiple of tape block size, for instance. Usually you need to write a program with specific knowledge of the particular device and data format to use an unbuffered device. Programs that require high speed or unusual tape formats use unbuffered devices. The commands **tp** and **debe** work best with unbuffered devices.

Buffered devices transfer data to or from buffers in the system disk-block cache, and copy data between the cache and the user program's buffers. Since the user's buffers have no restrictions on size or alignment, it's easy to handle devices with very general programs. However, the buffered device must have 512-byte records because that is the buffer size in the disk cache. Programs that want to read small or varying amounts of data at a time, without regard to physical block boundaries, use buffered devices. The commands **archive** and **debug** work best with buffered devices.

Some programs work with either buffered or unbuffered devices on tapes with 512-byte records because the programs read files in fixed 512-byte pieces. **hexodec** and **count** work with either device type.

Create these directories with the command line:
@>makedir /dev/mt /dev/rmt

The device name for half-inch tape using the NCR controller is either **cCdDB** and **cCdDBn** or **DB** and **DBn**:

C is controller number

D is drive number

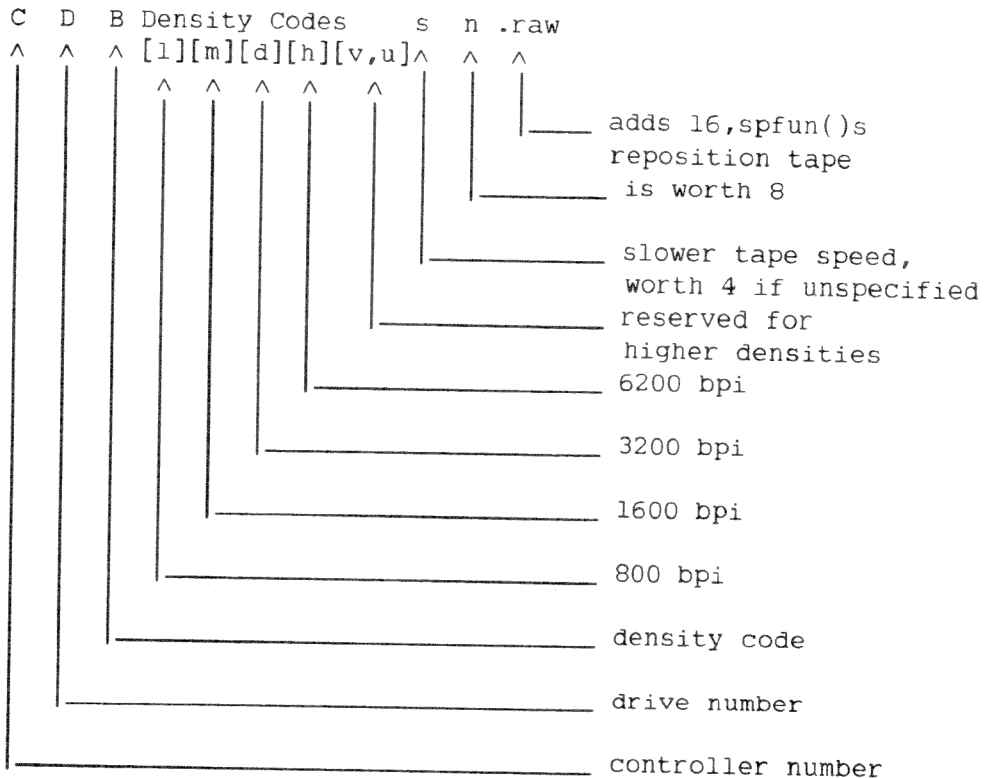
B is a density code that are

l	800 bpi	Not implemented on UNOS.
m	1600 bpi	
d	3200 bpi	Not UNIX.
h	6250 bpi	
v,u		Reserved for future higher densities.

n means don't rewind when the device is closed.

The figure below illustrates how devices are named.

Half-Inch Tape Device Names



This example displays half-inch NCR controller device names that include

- one drive, $D=0$, and the simpler name format
- the *B* code that adds 0 to 2 to the minor number for 1600-6250 bpi.
- *n* is worth 8
- a new attribute, *s*, meaning slower tape speed, that is worth 4 when unspecified
- *.raw* adds 16 and means that the driver is not to do any tape positioning automatically as the application will take over that chore, using *spfun()*s.

Half-Inch Tape Device

Major Device#	Minor Device#	Device Name b/[u]	Purpose
5	128	/dev/[r]mt/0ms	normal operation rewind to beginning of file
5	129	/dev/[r]mt/0ds	
5	130	/dev/[r]mt/0hs	
5	132	/dev/[r]mt/0m	
5	133	/dev/[r]mt/0d	
5	134	/dev/[r]mt/0h	
5	136	/dev/[r]mt/0msn	no rewind understands filesystem writes tape marks at file closing
5	137	/dev/[r]mt/0dsn	
5	138	/dev/[r]mt/0hsn	
5	140	/dev/[r]mt/0mn	
5	141	/dev/[r]mt/0dn	
5	142	/dev/[r]mt/0hn	
5	144	/dev/[r]mt/0ms.raw	no rewind use spfun() to rewind direct access to controller no concept of filesystem
5	145	/dev/[r]mt/0ds.raw	
5	146	/dev/[r]mt/0hs.raw	
5	148	/dev/[r]mt/0m.raw	
5	149	/dev/[r]mt/0d.raw	
5	150	/dev/[r]mt/0h.raw	

All the device nodes have to be specially marked to tell the pathname parser to pass any left-over pathname to the driver.

```
e>chmod =rwx,u+s /dev/emt/*
```

Minor device numbers for half-inch tape are decided in the following way:

Bits 0 and 1 are used together to indicate data density as follows

- 0 1600 BPI (bits per inch)
- 1 3200 BPI
- 2 6250 BPI (not tested yet)
- 3 reserved for future expansion

Bit 2 indicates tape speed. 0 is for lower speed; 1 is for higher speed. The drive used moves at 50 ips, inches per second, at 3200 or 6250 bpi, but the drive can move at either 25 or 100 ips at 1600 bpi. When this bit is 1, the tape controller uses an internal write-behind buffer to make continuous (streaming) writes possible. If you use the internal write-behind buffer, you do so at the cost of not knowing the amount of data correctly written when an error occurs.

Bit 3 is 0 if the tape rewinds when the program closes the file, and 1 if the tape is positioned so the next program to access the tape gets the next tape file after the current one. UNIX programs use this feature to select which file on a tape they read or write.

Bit 4 is 0 normally. When set at 0, the driver repositions the tape to find the data required by the program. When set at 1, the driver takes control of tape positioning and reads or writes at whatever is the current position.

For example, to back up onto 1/2" tape, type
package filename > /dev/rmt/0m

This command line copies the information in *filename* and returns to the beginning of the tape.

To retrieve this information, type
installation < rmt/0m

Chapter 2

Terminal, Printer, and Modem Connections

This section describes how to connect terminals, printers, and modems to your system. To connect these devices to your software, you must edit system files. The easiest way to edit files on UNOS is with the screen editor, **ted**. If you are unfamiliar with the **ted** editor, see the *UNOS Guide to Text Processing*.

Connecting to the Software

Once you have physically connected your terminal or terminals, you must logically connect them to the system. To do this you need to edit the file **/etc/ttys**. This file contains a list of all the ports on your system and designates them as either on or off. Ports 0, 1 and 2 is factory-configured as *on*.

To use the editor, **ted**, your system must know how your terminal does direct cursor addressing, scrolling and other functions. Before you can use **ted**, make sure the **/etc/termcap** file has an entry for your terminal type. The termcap file, by default, resides in **/etc/termcap**, however, the termcap file does not have to reside in **/etc/termcap**. To see if your terminal type is in **/etc/termcap**, type

```
@>p /etc/termcap
```

The termcap display includes all the terminal types your system supports.

The termcap file provides **ted** with a table of commands that describes your terminal's characteristics. We provide entries in the **/etc/termcap** file for frequently used terminals.

NOTE: All the terminal entries supplied have not been tested by Charles River Data Systems. Also, if you are using your terminal in a special manner, for example,

with 132 characters per line, refer to the **/etc/termcap** file for a complete description of the entry so that you can edit to reflect your usage. See *The Guide to Termcaps* for more information.

If your terminal is not on the list, refer to the section *The Guide to Termcaps* for information on how to create your own termcap description.

Terminals Listed in the Termcap File

Terminal Company	Entry Names
Ann Arbor	annarbor ambas
DEC	vi200 vt50 vt52 vt100
	vt132 decwriter I & II
General Terminal Corp.	sw80
HDS	avt c100
Hazeltine	h1500
IBM	ibm3101
Kimtron	kt7
Lear Siegler	adm3a adm42
Perkin Elmer	perkin elmer 1100 and 1200
Soroc	soroc
Teleray	t1061
Teletype Corp	tty33
Televideo	tvi912
Visual 50 Standard	vi50 vt52
Wyse 50	wyse50
Zenith	z19 z29 h19

If your terminal is in this list, simply follow the instructions below:
In the Command Shell, use the pound-sign command **#x TERM=** to
associate the terminal descriptor file with your terminal. The
command is

```
#x TERM=terminal_entry_name
```

Suppose your terminal is a Zenith z-19. You would type the
following command:

```
@>#x TERM=z19<cr>
```

In the Korn Shell, do not use the pound-sign command. In your
.profile file, type the command **TERM=terminal_entry_name**.
Suppose your terminal is a Zenith z-19, you would type
TERM=z19<cr>

This change is only in effect for the duration of your login.

To change your default terminal type permanently, edit the
/etc/ttys file. You can also change the designations of the other
ports on your system, if you have connected other ports.

Type the **ted** command and specify this file. You will see the
following list:

```
ADMIN>ted /etc/ttys<cr>
```

```
on:0:/dev/tty0:1/0:wyse50:speed=9600:::  
on:1:/dev/tty1:1/1:wyse50:speed=38400:::  
off:2:/dev/tty2:1/2:wyse50:speed=9600:::  
on:3:/dev/tty3:1/3:wyse50:speed=38400:::  
on:4:/dev/tty4:1/4:wyse50:speed=9600:::  
on:5:/dev/tty5:1/5:wyse50:speed=9600:::  
on:6:/dev/tty6:1/6:wyse50:speed=9600:::
```

Field

1. Labels the port as either on or off. Only login uses this field. If the field is *on*, then a login process will run on the associated terminal.
2. Refers to the port number. This number must be unique.
3. Refers to the device. For information about devices read the *Device Organization* chapter.

-
4. Refers to the Major/minor device numbers.
 5. Refers to your terminal type. Change the default, z19, to the name of the termcap entry that corresponds to your terminal.
 6. Lists ttymodes options separated by commas; they are set when login runs.
 7. Holds a space.
 8. Holds a space. Leave it empty for future additions.
 9. Tab over for an optional comment field. This field can have a user's login name, the name of your system's printer or the location of the device.

For example, to change the console (port 0) to a Zenith z-19, located in the computer room and used by the system administrator, type the following line:

```
on:0:/dev/tty0:1/0:z19:speed=9600:::<tab>admin
off:1:/dev/tty1:1/1:::
off:2:/dev/tty2:3/0:::
off:3:/dev/tty3:3/1:::
```

Installing Printers on Serial Ports

Like terminals, printers must be connected to the software. To connect a printer you must

- change the baud rate
- activate the login program
- identify device nodes with specific printers
- adjust software characteristics for your printer

You can create a printer daemon to queue and dequeue jobs to your printer. In your **/etc/startup** file, you can activate your printer functions when the system is in multi-user mode.

Change Baud Rate

After physically connecting your printer, you can change the baud rate with the **ttymodes** command. This command works the same for printers as it does for terminals. To set ttys to 1200 baud type

```
ADMIN>ttymodes speed=1200 < /dev/tty3
```

Activate login Program

You must edit the **/etc/ttys** file to ensure that the physical port, to which you connected the printer, has a word other than *on* in the first field of the port description. This field should only be *on* when a terminal is connected to the port. The *on* designation tells UNOS to run the **login** program on that device when the machine goes into multiuser mode.

A frequently used convention for this field is to identify the type of printer that is connected to the physical port. For example, if you connect the printer to serial port 3, the fourth line of the **/etc/ttys** file might read

```
on:0:/dev/tty0:1/0:z19:speed=9600:computer room::admin
off:1:/dev/tty1:1/1::speed=9600::
off:2:/dev/tty2:1/2::speed=9600::
```

```
print:3:/dev/tty6:3/4:default:speed=1200:1200 baud - spinwriter ::
```

Identify Device Nodes with Specific Printers

The **/etc/printers** file identifies device nodes with specific printers. Edit the **/etc/printers** file. This file should contain the names of all the printers on your system. Each line has three fields. The file can also have optional two-field lines.

```
ADMIN>ted /etc/printers <cr>
printer:/tmp/_queues/printer_queue:/dev/tty2
spinwriter:/tmp/_queues/spinwriter_queue:/dev/tty3
centronics:/tmp/_queues/centronics_queue:/dev/par0
default:printer
fancy:spinwriter
```


1. The first field is the name you've given your printer. All the printer commands use this name.
2. The second field is your printer queue's name. This should be a complete pathname. By convention, all printer queues have the suffix `_queue`. The printer daemon (the **lpd** command) creates and manages the printer queue.
3. The third field is the name of the device node associated with your printer.

The last two lines in this example refer to printer aliases. Commands that use printer names, such as **lpr** (enqueues line printer requests) and **lpq**, (displays the line printer queue) accept printer aliases.

Now confirm that your printer works by typing the following command line:

```
ADMIN>echo "this is a printer test" > /dev/tty3 <cr>
```

This command sends the string "this is a printer test" directly to your printer (port 3 /dev/tty3). If the string is not printed, check the following items:

- Is the cable connected from the correct port to the printer?
- Is the `/etc/ttys` entry for the physical port correct, that is, does the first field of the entry contain something other than `on`?
- Are any hardware or software switches on the printer defined differently than the characteristics in the **ttymodes** command? The key characteristics are usually baud rate, parity and word size.
- Does the device node exist?
- Are you redirecting the echo string to the correct printer port?

Adjust Software for Your Printer

Since different printers have different software characteristics, you may have to make some adjustments before your printer will operate correctly. The **ttymodes** command enables you to display and modify the software characteristics of the transmission to the printer. Check your printer's manual for its requirements.

The **ttymodes** command displays the default settings on your printer's port. To display these features, type the command and use, as standard input, the device name of your printer. For example, to check **/dev/tty3**:

```
ADMIN>ttymodes < /dev/tty3 <cr>
erase=^? linekill=^U retype=^R
stop=^S start=^Q kill=^C suspend=^Y eof=^D escape=^[
breaklist= ^wakealpha ^wakectl wakenl ^wakeblock
^i8bit prctl scrctl escape eof icrlf ealpha ectl
^erawctl etab ecrlf ^o8bit ^octl ocrlf oxtab
^irawedit ^enobellerr ^printing
ispeed=9600 ospeed=9600 wordlen=8 stopbits=1
^checkparity parity=none rts=on
```

Frequently, printer attributes are set with hardware switches. Use the **ttymodes** command to tell UNOS how you have set these switches. First check the parity and baud rate of your printer. UNOS sets the baud rate to the default speed of 9600 baud and the parity to none. Next, make sure that the input modes are the ones you need. In general, use the system defaults.

If you need to change any of these attributes, type the **ttymodes** command and specify the argument to the options that take equal signs (such as parity or speed) and turn attributes off or on by preceding the option with either an up-arrow (^) or nothing.

For example, the following command line turns off octl and oxtab and sets the parity to odd, and the word length to 7 bits on the printer associated with device **/dev/tty3**.

```
ADMIN>ttymodes ^octl ^oxtab parity=odd wordlen=7 </dev/tty3<cr>
```

Create a Line Printer Daemon Process

Once you can direct data to your printer, you can create a line printer daemon process for your printer. The **lpd** command is the printer daemon. It creates the printer queue and dequeues printing requests. Each printer on the system must have its own printer daemon.

The syntax of the command is
lpd printer.name

You can redirect standard error with the percent sign (%) so that printer error messages will not appear on the system console, but go instead to nowhere (**/dev/null**). You can also redirect standard out with the greater-than sign (>) so you can keep a log of the printer's activities.

Type the following command line to start the line printer daemon as a background process and redirect standard error and standard out.
ADMIN>/sys/lpd printer.name %/dev/null >/printer.log&<cr>

Activate Printer Functions in Multi-User Mode

Edit the **/etc/startup** file to ensure that every time you go into multi-user mode.

- you fix the line printer queue
- all your mode changes are in effect
- the line printer daemon for your printer starts
- all unfinished and waiting print requests queued prior to shutting down the system restart

The **/etc/startup** file is a script file, which is read and executed when the system goes from single user to multiuser mode. This file should contain the **lpmaint -fix** command, then the **lpd** command to start the printer daemon for your printer (run as a background process), and the **ttymodes** command if you've changed the default settings.

The following example shows the **/etc/startup** file configured for the spinwriter printer.

```
echo entering startup file
/sys/update &
delete /tmp/[^_]*
ttymodes speed=1200 < /dev/tty3
lpmaint -fix /tmp/_queues/spinwriter_queue
echo "/sys/lpd printer.name %/tmp/log.spinwriter &"
echo leaving startup file
```

Edit this file to corresponds to your system.

NOTE: The first time that you run the **lpmaint** command, the system will report errors because it is attempting to fix a nonexistent queue.

Using Modems

A *modem* is a device that allows you access to your system over telephone circuits. This section describes

- a simple set up for onsite and offsite modem connections
- dialing into onsite and offsite modems
- using an onsite modem with UniverseNet
- signing off
- general modem hints

Modem Types

This *Using Modem* section is only a guide. If you have an incompatible modem, see the instructions that accompany your modem. The information and examples in this section apply only to Hayes *Smartmodem* and compatible modems.

Most modems which support more than one speed will automatically detect the speed the caller uses, and adapt. You can connect modems to either an RS-422 or an RS-232C port on a Universe machine.

- The standard RS-422 (IOP/TP-308) connections on all Universe machines default for terminal use. Charles River machines support modems compatible with the Hayes *Smartmodem*, at speeds up to 2400 bps.

-
- The RS-232 interface on Universe machines will connect to modems of up to 19,200 bps.

Modem Set Up

This section includes instructions for setting up modems onsite and offsite.

Setting Up the Onsite Modem

Jumper the port connection used for either type of modem by **cross-connecting pins 2 and 3**. All other pins remain the same. Connect the selected port to the modem. Set the modem to auto-answer. Set the tty port to *on*. Users can now dial in and use the login program running on that port.

NOTE: Use a *null modem* cable from the port to the modem to avoid the jumpering step.

Setting Up the Offsite Modem

Follow the directions given in your modem's manual for physically connecting the offsite modem to the terminal. Make sure you use an appropriate cable (usually supplied with the modem).

Some modems allow you to connect your telephone to the modem so you can receive calls even though you are working on the terminal. See your modem manual.

Terminal defaults generally work correctly with modems, with one important exception. Change the baud rate to 1200 on your terminal, if it is not set there already.

Use upper case letters when you give commands to the modem. Once you start communicating with the system, change to lower case letters. The following examples reflect the use of upper and lower case as you should use them.

Dialing In

This section explains how to dial into onsite and offsite modems.

Dialing into the Onsite Modem

When connected properly, your terminal will display the prompt for which it is configured. Type

```
AT <cr>
```

Then, **OK** appears on your screen, with the cursor right below it.

NOTE: Some modems skip this step. Check with your system administrator.

The next line you type notifies the modem to dial the phone number:

```
ATDT[PHONENO]<cr>
```

Now type in the number. Note that the number will not echo on your screen. Check with your system administrator for the correct telephone number. Remember to use 1 and the area code, if necessary.

The modem responds with a **DIAL COMPLETE** message. On the next line, you receive either a **CONNECT** or a **CARRIER LOST** message. If your screen looks like the one below, you can continue.

```
AT
```

```
OK
```

```
ATDT[PHONENO]
```

```
DIAL COMPLETE
```

```
CONNECT
```

If you get the following screen, redial the number.

```
AT
```

```
OK
```

```
ATDT[PHONENO]
```

```
DIAL COMPLETE
```

```
CARRIER LOST
```

Using an Onsite Network

If your onsite system is connected in a UniverseNet network, you can use the Phase II networking commands to link from one system to another.

To sign onto a system other than the one the modem puts you in, use the **ttylink** command. See the *UniverseNet Administrators Guide* for more information. This is the sequence for successfully changing systems:

```
@>ttylink systemname <cr>
Trying . . . open
systemname TTYLINK server
```

If **ttylink** doesn't work, you will get this error message:

```
Trying...
ttylink:Access permission denied.
host systemname not responding
```

You can get an error message if your destination system

- Does not exist
- Requires access permission that you do not have
- Is not on the network

A successful **ttylink** connection brings up the **Name:** prompt on the destination system. You can then log in normally.

Signing Off

If you use **ttylink**, you must log off the system you are linked to, as well as the system you dialed into.

When you log off the system you entered with **ttylink** the following message displays:

```
<EOF>
host closed connection
```

When you log off the system you dialed into, you get the following display (and perhaps some garbage characters).

```
<EOF>
CARRIER LOST
```

You can dial back in to the home system at this point if you wish.

Working Hints

You can use all your terminal and system functions over the modem, as long as your remote login is configured correctly for your terminal.

Remember to save your work frequently. Sometimes your screen can freeze for no apparent reason, and the only way to unfreeze it is to reconnect to the system by shutting off the modem and terminal, and redialing. When you shut off the terminal and the modem, you lose all your unsaved work.

The Control S (^S)/Control Q (^Q) sequence may not always work correctly.

Working from a modem alters system responses. A fast typist can easily outdistance the modem's ability to return characters. Scrolling may take longer than you are used to.

Occasionally strange characters appear on the screen. Usually, these characters cause no harm to your work. However, if you have been particularly annoyed by recurring { characters every 20-30 seconds, the only solution seems to be disconnecting and redialing. Many application, such as **email**, react strangely to the { intrusion.

During bad weather, you may have trouble dialing into modems. On clear days, however, you should make a connection almost every time.

Chapter 3

User Accounts

This chapter describes user information for your Universe system. When you finish this chapter you can

- enter user names in a password file
- create user directories
- add a new user
- delete users

System Password File

Once you have decided how your disk is to be used, you can enter users' names in the password file and make user directories.

The password file, `/etc/passwd`, contains one single-line record for each user. Your initial password file contains only one entry--the system administrator's. The record is separated by colons (:) into fields:

```
admin::0:3:::/bin/command
```

A typical record might look like this [1]:

```
smith:aAlgbxxi:25:2::/usr/smith:/bin/command
```

A description and explanation of these fields follow.

User's Name (1)

This field contains the user's login name. It can be no more than eight characters or numbers. Make sure that you begin your name with a character, that is, do not create a name that is all numbers,

1. Case is significant--upper and lower case characters are different.

and do not use special characters, such as &, ! or @. The user types this word to login to the system. This is also the name by which the user is known on the system. The **email** and **write** commands use this name in order to know where to send their messages.

Encrypted Password (2)

This field contains the *encrypted* password that **login** uses to check against the user's typed-in password. If the field is empty, there is no password. When installing a new user, leave this field empty. If you have passwords on your system, ask the new user to create a password by using the command **cpw**. **Do not type the actual password in this field.**

User ID (3)

This field contains an identifying number associated with the user. While user id's are usually created sequentially, numbers can be reserved for special users such as a guest or the system administrator. Numbers must be unique. The largest user id that can be entered in the password file is 32767. **The system administrator's id is always zero.**

Group (4)

This field associates the user with a group. All users in a group share the same number. This number is referenced in the file **/etc/group**. Refer to the **GROUPS** section in this chapter for more information about groups.

Network Machine (5)

This field identifies the user's home machine for a network environment. For more information about this field refer to the *UniverseNet Network Administrator's Guide*.

User's Home Directory (6)

This field contains the name of the home directory for the user. While the name of the directory is commonly the same as the login name, it might be any directory name. Each user, however, should have a unique directory in order to have a personal mailbox.

Command Interpreter (7)

This field specifies the program run as the user's Command Interpreter. Usually the program is the UNOS Command Interpreter **/bin/command**, but the system administrator can select a different program if desired. For example, **/bin/basic** might be specified or if you have the UN/IX tool set, the Bourne or C-shell.

Groups

Groups permit restricted access to files based upon membership in a group. Members of a project can share protected files, or a database can run as a distinct group to access files.

NOTE: If you have networking, consult the *UniverseNet Network Administrator's Guide*.

A group has a unique name, number and optional password. This information is kept in the file **/etc/group**. No standard way of assigning groups, however, typically the system administrator assigns each project on a machine a group number. For example system sources and system maintenance are assigned different groups.

The format of the **/etc/group** file is as follows:
group_name:passwd:group number:list of permitted
users:comment

UNOS determines whether a user can access a file based on the three classes of permission associated with the file. The classes are *owner*, *group*, and *world*. Each class is further subdivided into read(r), write(w) and execute(x) permissions. The current permissions for a file can be seen by executing a **list** command.

```
$list -l filename.c
```

317 rw. ... r.x (1) jkw May 14 15:37:08 filename

The owner of the file has permission to read and write the file. Users in the Group have no permission to access the file and all other users can read and execute the file. The order in which UNOS determines which class of permissions to apply to a request is

1. owner
2. group
3. world

UNOS applies the first class of permissions that match the user requesting file access. This order implies certain restrictions. In the case of *filename*, the owner cannot execute the file, and users in the same group have no access permissions. All other users can read and execute the file.

Earlier versions of UNOS evaluated access permissions in the order of

1. owner
2. group
3. world

However, evaluation continued for each class if permission was denied in the current class. Files created under UNOS 4.0 and earlier had no group access permissions. These conditions create a potential access problem at UNOS 7.0. Files may exist with access permissions of "rwx ... rwx". In earlier UNOS versions, users in the same group as the owner had permission to access the file under the *world* class of permissions. In UNOS 7.0 these users are denied access to the files. To change the permission of the files use the **chmod** command with the *-groupfix* option. The format is

```
chmod -groupfix file1 file2 file...
```

This command line copies the world permission field to the group field.

Users

The *Users* section explains how to add a new user and how to delete a user.

Adding a New User

To add a new user to the system, have the user select a login name. Then use the screen editor, **ted**, to make an entry for the user in the system password file, **/etc/passwd**. For example, to add user **john**, type:

```
john::15:::/usr/john:/bin/command
```

Both **john** and the user id, **15**, must be unique.

Then use the **makedir** command to create the user's home directory.

```
ADMIN>makedir /usr/john
```

Use the **chown** command to change the owner of the directory to that of the new user, since the default owner of a directory is the creator--in this case **admin**.

```
ADMIN>chown john /usr/john
```

Deleting a User

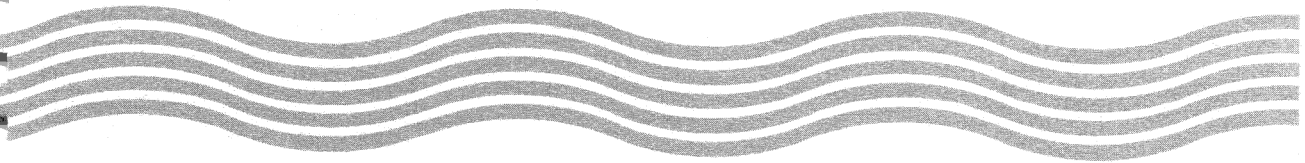
To delete a user from the system, use the editor to delete the user from the system password file, **/etc/passwd**. Use the **find** command to find all the files belonging to the user. Then use the **delete** command to delete the user's files and directories from the system. Make sure that you delete all the "." files, such as **.EDMAC** and **.globals**.

```
ADMIN>delete -r /usr/john/. * /usr/john/*
```

NOTE: Problems can arise if you delete the user, but not his or her files. This is especially true if you reassign the user id, because the system keeps track of users by id number and converts the number to a name by consulting the password file. Hence, reassigning the number also reassigns all extant files.

Section 2

System Administration



Contents

71	Chapter 1: UNOS File System
71	Structure of UNOS File System
73	File System Header
77	Storage
77	Contiguous Files
78	Features
80	Format
80	Limits
80	Disk allocation
81	Configuration File Variables
81	Types of Files
82	Data File
82	Directory
83	Device
83	Named Pipe
84	Eventcount
84	Multiple File Systems
85	File System Creation
86	File System Mounting
87	File Systems Unmounting
89	Chapter 2: The Configuration File
89	Configuration Parameters
97	The Configuration Table
98	Configuration Files Customization
98	Editing the Configuration File
101	Displaying the Configuration File
103	Example
105	Chapter 3: UNOS Priorities
106	The Way Priority Works on UNOS
106	An Example - ted
107	Priority Commands and Subroutines
107	Setting Priority
108	Displaying Priority
108	Priority Subroutines
109	Tuning Priorities
68	UNOS System Concepts

111	Chapter 4: Backup and Restore
111	Backup
111	package
114	debe
115	Considerations When Backing Up to Tape
116	Recommended Back Up Technique
116	Tape Partitions
117	Skipping Blocks When Copying
117	Tape Seeks
117	Buffer Sizes
117	New Tape Initialization
118	General Back Up Procedure
119	Administrator Functions For Back Ups
120	Automatic Back Up Script
121	Back Up Media Library and Log
122	Restore
122	installation
124	General Restore Procedure
124	Examples
124	Back Up and Restore the First Disk Partition
125	Back Up and Restore a Random Disk Partition
126	Store and Copy Tape Files
127	Back Up and Restore Selected Files
128	Back Up and Restore Directories

Figures

72	File System Layout
74	Small Files
75	Large Files
76	Huge Files
77	Gargantuan Files
83	Directory Data Block

Chapter 1

UNOS File System

This chapter provides an overview of the structure of the UNOS file system, explains the different types of files, and the division of files on the root directory.

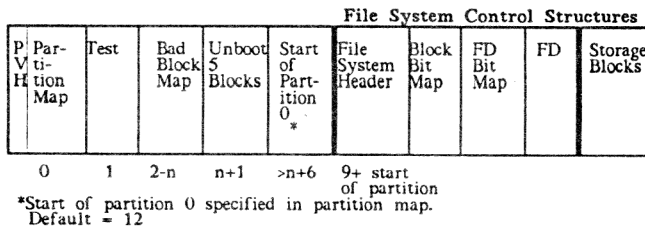
Structure of UNOS File System

A *file system* is a complete directory structure; this includes the root and its subdirectories. A file system exists on a partition, which is associated with some form of magnetic media. A file system can be any size. A floppy disk could hold a file system of 500-1,000 blocks, while a large disk could have a single file system (encompassing the entire disk), or be divided to hold several smaller file systems.

A logical file system is one or more separate file systems (one per partition) logically joined into a single file system. A file system is only recognized by UNOS after it has been mounted. *Mounting* means that UNOS has made the logical connection between the file system on the partition and the root file system. You must mount each file system separately. For more information about mounting, refer to the section *File System Mounting* in this chapter.

Every file system has the same general format. The first nine blocks on the device are reserved for system information. The following diagram shows what these nine blocks contain.

File System Layout



The actual data in the file system begins at block nine. Although all file system layouts are the same, they differ slightly in the amount of the space needed for bit maps and file descriptors (FD), depending on the number of blocks the file system manages.

See the *Disk Configuration* section of *System Set Up* in this manual for a complete explanation of disk configuration. Descriptions for the commands needed for configuration are in the *UNOS Commands Manual*: **copy_boot**, **fscheck**, **makefs**, **mount**, **write_bad_blocks**, **write_partition**, and **unmount** or **umount**

Volume ID

Contains the location and length of the bootstrap program and a character string identifying it as a volume id block. It also is the location of the partition record.

Test

Marks a file system as mounted. On all systems the **mount** and **unmount** commands use this area.

Bad Tracks

Contains the location of the bad tracks on the disk.

Boot Strap

It contains the UNOS boot strap program.

Bad Tracks, Boot Strap, Test and Volume ID are only used if the file system is physically located at the beginning of the disk. If the file system is not physically at the beginning of the disk, these blocks are unused. For a more complete description of the PVH, bad block mapping, and the UNOS boot strap program, see *Disk Organization* in the *System Set Up* section of this manual.

File System Header

This area describes the size and location of the Bit Map, FD Bit Map, File Desc, and Storage.

Bit Map

This is a map that contains a bit for each block in the file system. Each bit is marked as either 0 for unused or 1 for used. The bit map is updated every 30 seconds when the **update** command executes the **sync()** subroutine and when more of the bit map is read into memory. The **space** command looks at this map to find out the number of free blocks.

FD Bit Map

FD Bit Map contains a bit for each file descriptor, marking it as either 0 for unused or 1 for used. This map is updated every 30 seconds when the **update** command executes the **sync()** subroutine and when more of the FD Bit Map is read into memory.

File Desc

One file descriptor exists for every file the system can have. Unless otherwise specified (to the **makefs** command), the system allocates 9/64 the number of blocks for file descriptors--this is roughly one file descriptor for every eight disk blocks. A single file system can contain up to 32,766 file descriptors. File descriptors control access to files.

A file descriptor contains information about the file it describes, such as

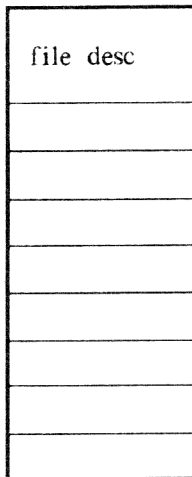
- the file type (data, device, or eventcount)
- the permission modes on the file
- the number of links and various control bits. For data files, it contains information about the size of the file. There are four file sizes:

-
1. *small* 0-4096 bytes
 2. *large* 4096-524288 bytes
 3. *huge* up to 67 million bytes
 4. *gargantuan* up to 2,147,483,647 bytes

The file descriptor also contains an array of eight elements that contain pointers to the file. These eight elements actually point to eight blocks on the disk (each block is 512 bytes in size).

The file descriptor alone can handle a file up to
 $512 * 8 = 4096$ bytes

Small Files

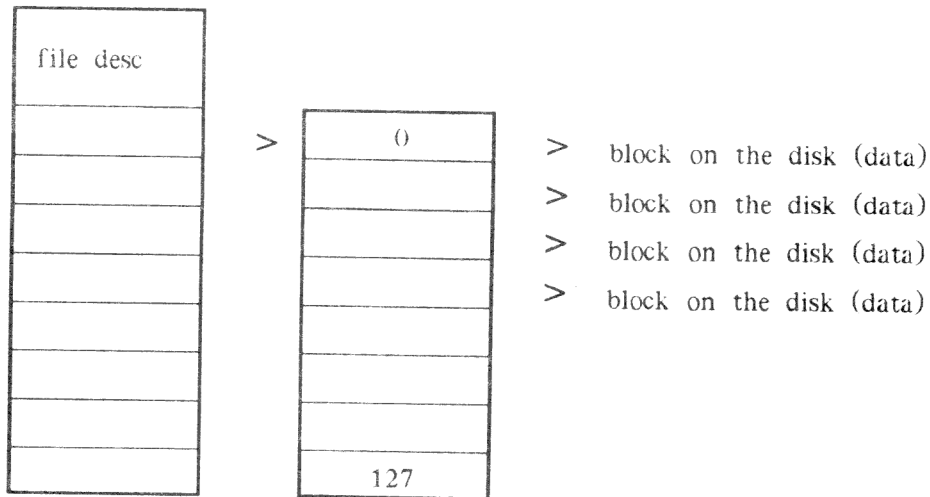


- > block on the disk (data)
- > block on the disk (data)
- > block on the disk (data)
- > block on the disk (data)

4096 bytes

If the file is larger than the small file, the first eight pointers are copied to an indirect block and the first element in the file descriptor points to this block.

Large Files



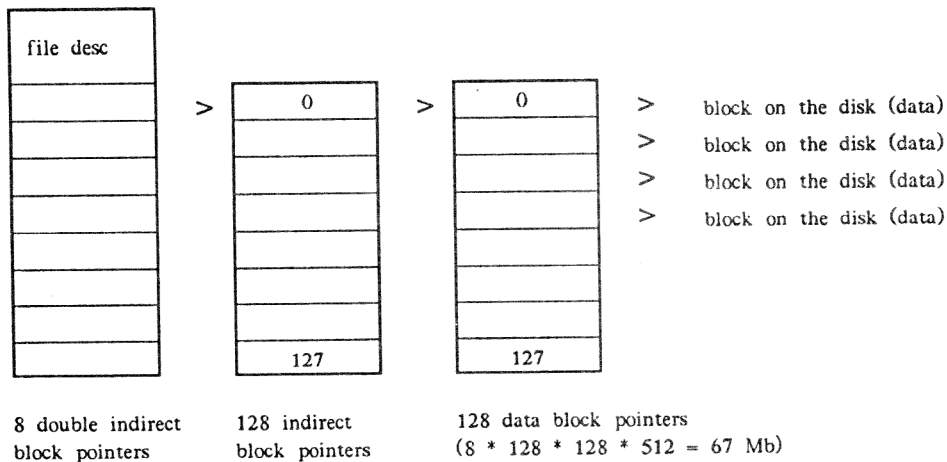
8 indirect
block pointers

total of 8 tables
 $(8 * 128 * 512 = 524,288 \text{ bytes})$
128 data block pointers

A large file uses indirect referencing to locate its blocks on the disk. This indirect block contains up to 128 elements, which points to a maximum of 128 blocks. The original file descriptor array points to the indirect array, which points to the actual disk blocks. On this level, you can have 8 of these 128-element indirect tables. Therefore, the file descriptor's eight elements would all contain pointers to these tables. For this reason, I/O in larger files can be slower requiring as many as two disk accesses to read the file's data. The disk cache, however, keeps indirect blocks in memory, and thus, large files are accessed more quickly.

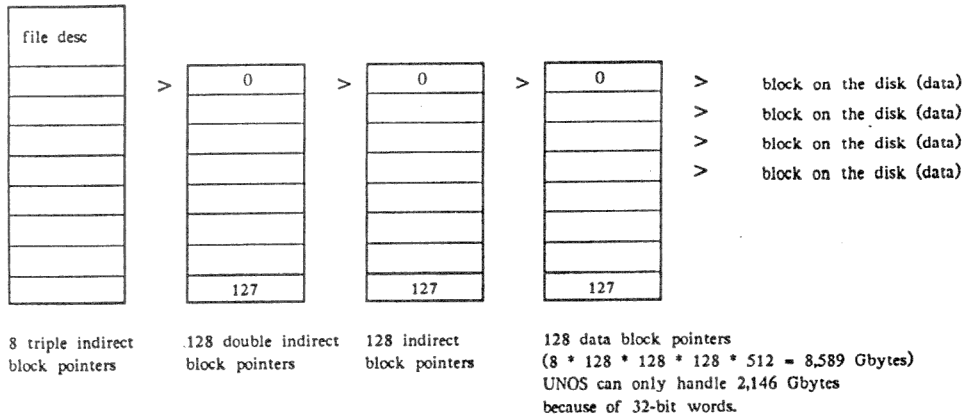
A huge file uses double indirect referencing; that is, the first 128 blocks are written to the second indirect table, which then points to blocks on the disk. There can be 128×8 such pointers. Thus, it could require three reads to access the disk.

Huge Files



A gargantuan file uses triple indirect referencing; that is, the first 128 blocks are written to the second indirect table, which then points to a third indirect table. There can be $8 \times 8 \times 128$ such pointers. Thus, it could require four reads to access the disk.

Gargantuan Files



Storage

Storage stores data. This is the amount of free blocks available after subtracting the file system control structures from the original size of the file system.

Contiguous Files

Contiguous files consist of contiguous areas of disk called extents. *Extents* are contiguous chunks which can be from one block to two giga blocks long. A contiguous file can contain from 1 to 786432 extents. Use the **contig** command [1] or the **open()** subroutine [2] to create contiguous files. In either case, you can specify an acceptable minimum size for each extent of the file. A contiguous file can

1. See the *UNOS Commands Manual* for more information on the **contig** command
2. See the *UNOS Subroutine Manual* for more information on the **open()** subroutine

keep on growing indefinitely, although, if it goes over three extents, it needs an indirect block.

A set of **fcntl()** operations for contiguous files

- sets and reads the current extent sizes
- gets the total current allocation size (as opposed to the file size)
- forces a new allocation and gets the size of the largest free extent on a specified partition

Features

Contiguous files provide these features:

1. Maximum and minimum extent sizes can be specified on a per file basis.
Allocation of new extents can be prevented on a per file basis by setting the maximum extent size to zero. This is the *-freeze* parameter for the **contig** command.
Allocation parameters can be modified as often as necessary.
System default allocation is indicated when the user allocation values are both -1.
2. The maximum number of extents is 786,432.
3. An extent may be from 1 to 4,194,304 blocks long. Actually, an extent could be as long as 2,147,483,647 blocks long, but only the first 4,194,304 blocks could be addressed with a positive 32-bit file position.
4. Determination of the position of a block in a file is proportional to the number of levels of indirection, rather than the number of extents (which is the more usual case).
Redundant length information in the file allows this performance enhancement.
5. The large size of extents and the associated fast access avoids the performance problems created by highly fragmented disks.

NOTE: A large write to a contiguous file can cause allocation of an extent larger than the specified extent

size, to a maximum size equal to the size of the write.

Contiguous files have the following advantages over block files:

Reduced mapping time

Contiguous files with extents whose size is large (say, greater than 128 blocks) have an advantage over block files, because the mapping time is either greatly reduced or eliminated. Mapping time is the time it takes the operating system to convert a logical block number to a physical block number. It is related to the number of levels of indirection that the mapping routines must go through. As an example, a 2000-block (1 MB) block file would require two levels of indirection. A 2000-block single extent contiguous file would not require any.

Reduced seek time

Seek time is less because the blocks in an extent are closer together. Access to nearby logical locations results in smaller disk arm movements.

Reduced transfer times

Large transfers tend to use direct (raw I/O) transfers. Small transfers tend to use the multiblock buffer. Consequently, the physical transfer time is minimized.

Contiguous files access more efficiently

Finding the final disk address of a transfer requires no indirect blocks to go through. Reading and writing a multi-block piece is done with a single I/O request to the driver (ordinary files might require multiple transfers because of random block allocation)

NOTE: Both pointers to indirect blocks and pointers to extents are associated with a length. This length specifies the total size of the extent(s) reachable through this pointer. The extent allocator uses a conventional block bit map with state variables to speed up processing.

Format

The file descriptor consists of 3 extent address-length pairs and two user-specified allocation parameters. Thus, up to 3 extents may be referenced directly. Up to 3 levels of indirection are supported. The **Limits** section describes the extents and levels of indirection for contiguous files.

Limits

The CSMALL contiguous file can directly address 3 extents. These extents may be of any size and at any location.

The CLARGE contiguous file can address 192 extents with one level of indirection. Access to a particular block in the file requires, at most, access to one indirect extent block.

The CHUGE contiguous file can address up to 12,288 extents with two levels of indirection. Access to a particular block in the file requires, at most, access to one double indirect extent block and one indirect extent block.

The CGARGANTUAN contiguous file can address up to 786,432 extents with three levels of indirection. Access to a particular block in the file requires, at most, access to one triple indirect extent block, one double indirect extent block and one indirect extent block.

Disk allocation

UNOS maintains the free space bit map. This bit map has a one-to-one correspondence between the blocks on the file system. Allocation of free blocks follow these rules:

1. Blocks are always allocated from the beginning of the disk on a first fit basis. Thus, when n blocks are requested, the first n contiguous blocks are allocated, regardless of whether there is a better fit somewhere else on the disk.
2. When a request of n blocks is made and n blocks are not available, the largest area available is allocated. The caller is

informed of this fact. For example, if 100 blocks were requested and only 1 block is free, then this 1 block is allocated. When areas larger than the largest available area are requested, the location and length of the most recent largest area is saved so that subsequent requests are treated more efficiently. By doing this, an upper bound on the size and a lower bound on the location of the next largest area are obtained.

Configuration File Variables

The following configuration file variables affect contiguous allocation:

typical value	config file variable	Usage
0	cnf__contig	Specifies whether the system creates contiguous files by default.
1	cnf__cmin	Specifies the smallest extent. The default is one (1) block.
8	cnf__cmax	Specifies the largest extent. The default is eight (8) blocks.

Refer to the *The Standard Configuration File* chapter in this section for descriptions of these structures.

Types of Files

Files can be of five types:

1. a plain data file
2. a directory
3. a device file
4. an eventcount
5. a named pipe.

The **list -l** command displays these files with special letters. The following example is

- a listing of a data file, *index*
- a directory, *new.install*
- a device, *cfs0*
- an eventcount, *clock*
- a named pipe, *(/dev/Fred)*

Notice that the first field indicates what kind of file it is. The absence of a letter means it is a regular file, the letter **D** means that it is a directory, the letter **B** or **U** means that it is a device, the word **EVENT** means that it is an eventcount, and the word **Pipe** means a named pipe.

	1408	rw.	...	rw.	(1)	dianne	Sep 21 82 13:25	index
D	1632	rw.	...	rw.	(1)	dianne	May 4 12:02:35	new.install
B	4, 98	rw.	...	rw.	(1)	admin	Apr 25 16:31:09	cfs0
EVENT	1	rw.	...	rw.	(1)	admin	Apr 25 16:31:09	clock
PIPE	0	rw.	rw.	rw.	(1)	john	Sep 17 15:18:15	/dev/Fred

Data File

A data file is what is typically known as a file. Its data blocks can contain any type of data: ASCII text, binary data, or an executable program. The file system does not differentiate among these files.

Directory

A directory is just a data file. The only difference is that it is marked as a directory, and consequently users cannot write to it directly. Its data blocks contain a table of file names and file descriptor numbers. These names can be up to thirty characters, and the numbers are two bytes. These numbers point to file descriptors on the file system. The number -1 or 0 means that the space is unused.

Directory Data Block

30

2 bytes

entry name	file#
entry name	file#

The root directory is always FD number 0 and the number 1 is reserved. The word *root* here means root of this file system -- which may or not be the root of the entire logical file system.

Device

A device uses only a single file descriptor, and therefore, requires no data blocks. Its file descriptor contains the device's major and minor device numbers. A device is designated as buffered (B) or unbuffered (U). These names refer to the way I/O is handled. A buffered device collects data in 512-byte buffers and when the buffer is full, writes it out to the disk. An unbuffered or raw device writes directly to the disk. Most devices can be accessed with either mode. For more information about devices, refer to the *Device Organization* chapter in the *System Set Up* section.

Named Pipe

Named pipes provide a pipe-like interprocess communication mechanism for processes which are not related as parent-and-child. Typically, one process (like a printer daemon) will read the pipe continuously, and other processes open it for writing and place requests into it. Named pipes have all the attributes of pipes, plus the advantage that they are universally accessible.

A named pipe is a single file descriptor, plus the required data blocks. Named pipes can be files of type SMALL, or any contiguous type (CSMALL through CHUGE). There is a limit, however, of 8 blocks of unread data in a pipe.

File System Mounting

You incorporate a file system into the logical root file system by mounting it on some existing directory. You can use any directory -- on the root file system or on an already mounted file system. Only one file system can be mounted on one directory. Once you have mounted a file system, the pathname to the directory refers to the root directory of the mounted file system and assumes all the attributes of that directory; that is, the owner, access modes and so on are the same. Any files in the original directory are hidden and are inaccessible while the file system is mounted on it. These files will reappear when the file system is unmounted.

The **mount** command actually mounts a file system.

NOTE: Even if your physical disk contains several file systems, each one must be mounted separately.

Since the **mount** command associates the particular file system with a logical device, a file system can be independent of its mounted directory and mounted to a different one if necessary. For example, you could mount your floppy disk on the directory **/mnt** (by convention, the directory reserved for mounting temporary devices) or on another directory.

The syntax of the **mount** command is as follows:

mount device directory

In our case, we will mount the floppy device onto the holding directory:

```
@>mount /dev/cfs0 /mnt
```

Both the device (with a valid file system) and the directory must exist and you must have read and write access to both of them. If you want to mount your device on a directory other than **/mnt**, create it first before using the **mount** command.

```
@>makedir /special.dir
```

```
@>mount /dev/cfs0 /special.dir
```

You now have connected your file system to the rest of UNOS. You can use any of the system commands on it; this means that you can copy files to and from it, change to it, create subdirectories and so on.

If you would like to know what file systems are mounted, type the **mount** command without an argument and you will see what devices have been mounted by the **mount** command. (The root does not appear in this listing, because, although it is a mounted device, it was not mounted by the **mount** command.)

Once a device is mounted on a directory, any files in the directory are hidden. In the following example, we type the **mount** command alone to see what directories have been mounted. Then we do a listing of the **/mnt** directory. We mount the floppy **/dev/cfs0** as **/mnt** and do a listing of its contents. Notice that the three files **list**, **log**, and **tmp** are no longer visible.

```
@>mount
```

```
@>list /mnt
```

```
/mnt:
```

```
list
```

```
log
```

```
tmp
```

```
@>mount /dev/cfs0 /mnt
```

```
@>list /mnt
```

```
/mnt:
```

NOTE: Make sure that the permission modes of the mounting directory match those of the users attempting to mount it; otherwise they cannot access their files.

File System Unmounting

When you are finished using your floppy, you can't just remove it from the floppy drive. You must tell the system that you want to unmount it. Use the **unmount** command.

NOTE: If a mounted file system has any open files, or is being used as any one's current working directory, the file system can not be unmounted. (For more information about mounting and unmounting file systems, refer to the *Automatic Power Up and File System Checking* section of the *Universe System Installation Manual*.)

The command has the following syntax{4}
umount directory or device

Referring to the previous example, you could unmount either the device **/dev/cfs0** or the directory **/mnt**.

When you get the system prompt back, you can safely remove the floppy from the drive.

4. In a UNIX environment use the **umount** command, and you must be superuser.

Chapter 2

The Standard Configuration File

Configuration files let you specify various system parameters, such as the number of system buffers, the number of logical units (lu) and the number of open files. The configuration file provided with each release of UNOS describes a general purpose system that balances the needs of CPU-bound programs with those of I/O-bound programs.

Configuration Parameters

The following section describes each of the adjustable parameters and then follows with the actual file, `/etc/conf.c`.

cnf_rootdev -- root device

The configuration file expects the Major/minor numbers of the root device. These are the high and low bytes of a word. $4*256 + 2$ refers to dev 4/2 or 0402. The root device is where the logical file system root resides and is often different from the boot device specified when the system was booted. For example, in the UNIVERSE 68/35 system, cq0 is the boot device, while cq2 is the root device. Refer to the *UNOS Device Table* in the *Reference* section of this manual for a listing of the boot and the root devices.

cnf_swapdev -- swap device

This is the pathname of the swap device. It must exist on the just specified root device and must be a device-type file.

cnf_swapsize -- swap device size

The configuration file expects the physical size of the swap device in system blocks (512-bytes). To disable swapping, make this number zero.

cnf_nsysbuf -- number of system buffers

The kernel uses the system buffers to cache disk blocks on buffered

I/O. In order to do this, system buffers store information about the I/O request (in the control block) and hold data (in 512-byte buffers) when reading and writing to the disk. This is how read-ahead for disk I/O is implemented. When the system notices that a second sequential block from a file is requested, it starts I/O on the third. When the third is requested, it will probably be in memory as a result of the previous read-ahead, and thus, the system can start I/O on the fourth.

As blocks are read into the cache, their disk location is noted. This is used to allow the cached block to be subsequently used instead of the actual disk block. If your system has a lot of disk I/O, many system buffers will make I/O faster. There is a trade-off, however, since each buffer requires more than 512 bytes of memory, which cuts into the amount of usable memory. Also, the more system buffers you have, the longer it takes to search through them.

The system writes out the disk buffers in the following instances:

1. Every 30 seconds in multiuser mode, the background process **update** writes out any modified buffers. On a system basis, this also can be done anytime with the **sync()** subroutine.
2. When you use the **unmount** command to unmount a file system, the **shutdown** command to shutdown the system, the **space** command to find out the number of free blocks on the system, or the **fscheck** command to check the file system.
3. To make room for new blocks in the cache, a least-recently-used algorithm chooses the oldest block in the buffer to write out.

cnf_numproc -- number of process control blocks

This is the number of processes that can be active at one time. 30 processes is the standard number for systems with 1 Mbyte or more of memory.

NOTE: The entries "number of process control blocks" and "number of different shared segments" can be no greater than 255. The values of the other entries, however, are determined by the sizes of the corresponding fields in the structure `Whose` whose sizes are defined in the file `/include/sys/conf.h`.

cnf_start_ceil_pri -- ceiling/floor for start process

This is the ceiling and floor priority for the start process and for all its children; thus, it is the priority level for all the processes on the system.

cnf_swap_priority -- ceiling/floor for swapper process

The swapper is set at a fixed priority that is higher than any other process on the system, to ensure that it can always swap out other processes. **Do not change this number.**

cnf_filler1 -- fill1

This is a place holder in the structure. It is read as fill one.

cnf_maxlu -- number of lus

Logical units are used to keep track of the number of processes that have open files. When a process wants to open a file, it finds a free *lu* entry in the *lu table*. It uses that *lu* to store a pointer to a file descriptor in the file descriptor table and to record its own position in the file. Consequently, two processes independently opening the same file (or even the same process opening the same file twice) will each have a unique *lu* entry, since their position in the file can be different. The *lu* pointers, however, will point to the same file descriptor.

You can specify the maximum number of logical units (*lus*) in the *lu* table. If the system runs out of *lu* entries before file descriptors, it will generate the error, ENOLU (-9). This could occur if there are many processes opening files. For a listing of the error messages, refer to Appendix C in your respective *Universe System Installation Manual*.

cnf_nfopen -- number of open files

This is the number of physical files of all types (including eventcounts) that can be open at one time. This number is used to reserve memory space for the file descriptors. There is one file descriptor for each unique open file on the system. If you attempt to open more files than there are file descriptors, the system will generate the error ENOFD (-8), which means that there are no more free file descriptors slots.

NOTE: Since the system only uses one file descriptor per file, the number of available file descriptors is independent of the number of processes running.

For a list of the error messages, refer to Appendix C in your respective *Universe System Installation Manual*.

cnf_neventcalls -- number of ec calls

A process can not wait for more than three simultaneous events with the eventcount subroutine **ecwait()**. It can, however, wait for an unlimited number of pending asynchronous events with the eventcount subroutine **ecall()**.

This number specifies the total number of outstanding **ecall()** routines at one time for all processes in the system.

cnf_ndsegs -- number of different shared segments

This number specifies the number of distinct segments (text and data) that can be shared. The system keeps track of these shared segments in a table whose total number of entries is specified by the configuration file. When the last process sharing a segment terminates, the system will then free up that slot in the shared memory table and allow another segment to be shared.

cnf_iawait_pri factor -- interaction credit

The I/O wait factor is one of the process scheduler's tuning parameters. When a process waits for a child, an eventcount, a terminal to complete output or the **sleep** command, its subsequent priority is increased and its time slice (quantum) decreased. The result is that interactive processes run more frequently (increased priority), but for a shorter time (decreased quantum).

Making this number larger (1024 maximum) would result in changing the time needed to recover from an I/O interaction. The larger the number, the faster the recovery. If it were set at the maximum, the Command Interpreter would always run at 100.

cnf_quantum_pri_factor -- quantum used up demerit

When a process uses up its time slice, the system decreases its priority by this amount. As a process uses up its quantum, it is assigned a lower priority and a larger quantum. The swapper expects this number to be eight. If you were to change it, the

swapper would misjudge when to swap out a process.

cnf_ttybufsize -- size of buffers per tty

Each teletype has a default 256-character input buffer. This number specifies the input tty buffer size for each tty on the system. If the value is not a power of two, between 2 and 32768, it defaults to a power of two. The output buffer size is fixed at 256. For block mode I/O, increase the buffer size to 2K to 4K per tty. When changing the buffer size, take into account the amount of memory on your system, since *all* terminals will be allocated the same size buffer independent of their on/off status.

cnf_nfsys -- number of mounted file systems

This number specifies the total number of simultaneously mounted file systems. Mounted file systems allow you to logically connect the files on another device, such as a disk cartridge or floppy, into the logical file system. This number can never be less than one, otherwise the system can not mount the root at boot time.

cnf_logsize -- log area size in bytes

The system has an experimental feature that allows taking a histogram snapshot of the system and user processes to help determine performance bottlenecks. This feature is currently supported by the **profile** command. A suggested size of the log area is 1024 bytes.

cnf_clock_rate -- clock chip rate

This corresponds to the clock rate on your CPU board not the calendar clock. **Do not change this value.**

cnf_pause_at_boot -- wait before mounting root

(A system test feature.)

cnf_filler3 -- floppy step time

(Unused)

cnf_nlocks -- maximum number of file locks

The system allows locking arbitrary areas of files for exclusive access. This number specifies the total number of simultaneous file locks in the system. The recommended number is 40.

initialization no additional memory is available. You should configure a minimum of 200,000 bytes for the network. For more information about this parameter, refer to the *UniverseNet Network Administrator's Guide*.

cnf_ipcnques -- number of ipc queues

This number specifies the number of IPC queue structures to allocate in the kernel. The default value is 50. If you try to create more queues than are specified by this entry, you will receive the **ENOQSPACE** error.

cnf_ipctotmsg -- total number of message bytes

This number specifies the area which is used to hold messages on all the queues. The default value is 1000 (bytes). Each enqueued message requires <size of message (multiple of 2 bytes)> + 30 bytes. If you try to send a message requiring more bytes than are available, you will get the **EQFULL** error, if the **IPC_FNOWAIT** flag is set in the flags argument to the **ipcsend()** routine.

cnf_ipcmsgsiz -- max size of a single message

This number specifies the maximum size of a single message. The default value is 8192 (bytes). If you try to send a message larger than that allowed by this field, you will get the **EPKT2BIG** error.

cnf_ipcqsize -- max message bytes on one queue

This number specifies the total number of message bytes allowed on a single queue. The default value is 16384. If you try to send a message that would increase the number of bytes on the queue to greater than specified by this field, you will get the **EQFULL** error if the **IPC_FNOWAIT** flag is set.

cnf_contig -- create contiguous files

This field specifies whether the system creates contiguous files by default. Set this field to zero.

cnf_balloc -- allocation size

This field specifies the initial file chunk. The values are one (1), meaning one block, to eight (8), meaning eight blocks at a time. Do not use eight for a normal file system, that is, one with a variety of file sizes, since it can be very wasteful of space.

cnf_cmin -- minimum contiguous size

This number specifies the smallest extent. The default is one (1) block.

cnf_cmax -- maximum contiguous size

This number specifies the largest extent. The default is eight (8) blocks.

cnf_dfskill -- distributed file system requests

This field enables or disables the distributed file system request. The default is *No*.

The Configuration Table

This configuration table is an example; your table may differ. Later in this chapter, the *Configuration Files Customization* section explains how and why to change your configuration file.

Current configuration from Memory

ROOT device is 4, 10.
Swap device is '/dev/swap'.
Swap area is 8000 blocks.
80 system buffers allocated.
100 process control blocks allocated.
Ceiling priority is 100 for start process.
Floor priority is 32 for start process.
Swapper priority is 200.
Maximum logical units is 400.
Maximum number of open files is 200.
Maximum number of event calls is 100.
Maximum number of shared data segments is 200.
Priority tuning interaction credit is 512.
Priority tuning disk wait credit is 0.
Priority tuning quantum used up demerit is 8.
Input buffers are 256 bytes per tty.
Maximum mounted file systems is 8.
Histogram log area is 2000 bytes.
Clock chip rate is 1250000 Hertz.
Will not wait before mounting ROOT.
Maximum number of locks (lock lists) is 40.

Default erase character is '\177'.
Default line kill character is '\25'.
System name is ''.
5 blocks in multi block I/O buffer.
1024 bytes in system error log buffer.
Recovery ROOT device is 4, 8.
Script is '/etc/startup'.
Auto start.
Mount ROOT read/write.
Number of pseudo teletype devices is 2.
Poll after 20 services.
Distributed systems structures memory is 0.
Number of ipc queues is 0.
Total number of ipc message bytes is 0.
Maximum size of a single ipc message is 0.
Maximum ipc message bytes on one queue is 0.
All created files are not contiguous.
Non contiguous file allocation chunk size is 0 blocks.
Default minimum contiguous chunk size is 0 blocks.
Default maximum contiguous chunk size is 0 blocks.
Outgoing distributed file system requests are disabled.

End of Configuration from Memory

Configuration Files Customization

You can create an alternative configuration file customized to the specific needs of your system. This section describes how to change your configuration file with the **confedit** command. Also included is an explanation of how to display the contents of your configuration file with the **conftable** command. The *Example* section describes changing the number of system buffers in the configuration file to improve I/O performance.

Editing the Configuration File

The **confedit** command creates new configuration files and alters existing files. To use the **confedit** command, you must be the system administrator.

When you edit the configuration file on the main root, also change the configuration file on the back-up root. The back-up root loads the configuration file at boot time. If you didn't change the configuration file on the back-up root to match the one on the main root, the changes will not appear when you reboot.

confedit looks for the configuration file named. If the file exists, **confedit** opens and checks for correct configuration file format. An error is reported if the file is not in the correct format. If the file does not exist, **confedit** creates it.

Assignment statements are read from stdin in the -v or non-interactive mode and parameters change accordingly. The interactive mode prompts the parameters one at a time as you enter the desired values. Interactive mode is the default. If you do not want to change the parameter value, type a <cr>. To return to an item, type **Shift 6** which appears as ^. This key sequence moves you backwards one question at a time. An example of interactive mode follows. Your responses are to the right of the equal sign (=).

ADMIN>confedit conf

ROOT device (0, 0 means ask). [4, 10] =
Swap device (NULL means ask). ["/dev/swap"] = "/dev/swap2"
Swap device size in blocks. [8000] = ^
Swap device (NULL means ask). ["/dev/swap2"] =
Swap device size in blocks. [8000] = 8000
Number of system buffers. [80] = 80
Number of process control blocks. [100] = 100
Ceiling priority for start process. [100] = 100
Floor priority for start process. [32] = .
Priority for swapper process. [200] = 200
Number of logical units for whole system. [400] =
Number of open files for whole system. [200] =
Number of event calls for whole system. [100] =
Number of (shared) data segments. [200] =
Priority tuning interaction credit. [512] =
Priority tuning disk wait credit. [0] =
Priority tuning quantum used up demerit. [8] =
Bytes of input buffer per tty. [256] =
Maximum mounted file systems. [8] =

Size (in bytes) of histogram log area. [2000] =
 Clock chip input rate in Hertz. [1250000] =
 Should wait before mounting root? (y/n). [No] =
 Maximum number of file locks (# locklists). [40] =
 Default erase character. ['\177'] =
 Default line kill character. ['\25'] =
 Name of computer system. [""] =
 Blocks in multi block I/O buffer. [5] =
 Bytes of system error log buffer. [1024] =
 Recovery ROOT ('0, 0' means ask). [4, 8] =
 Name of script. [""] =
 Auto start? (y/n). [Yes] =
 Mount ROOT read-only? (i.e. tape) (y/n). [No] =
 Number of pseudo teletype devices. [2] =
 Poll after this many services. [20] =
 Amount of memory to grab for distributed systems structures. [0] =
 Number of ipc queues. [0] =
 Total number of message bytes. [0] =
 Maximum size of a single ipc message. [0] =
 Maximum message bytes on one ipc queue. [0] =
 All created files are contiguous? (y/n). [No] =
 Allocation chunk size in blocks. [0] =
 Minimum allocation size for contiguous files in blocks. [0] =
 Maximum allocation size for contiguous files in blocks. [0] =
 Disable outgoing Distributed file system requests? (y/n). [No] =
 ROOT device (0, 0 means ask). [4, 10] =
 Swap device (NULL means ask). ["/dev/swap"] = done

You can create a batch file to change one or more options in one or more configuration files. Include only the features you want to change in a file. For example, create a file, *cmds*, with these items:

```

iop_pollrate = 100
dfs_kill = y
  
```

A complete list of options appears in the *Configuration Parameter* section of this chapter. All parameters begin with a *cnf__* prefix. You do not need to use the *cnf__* prefix when identifying parameters in a batch mode change file. However, using the prefix will also work.

Type this command line to institute the changes in the *cmds* file in the configuration file, *conf*.

ADMIN>confedit < cmds conf

If you want to see the changes display as they occur type

ADMIN>confedit -v cmds conf

Poll after this many services. Was:[20] Now is[100]

Disable outgoing Distributed file system requests?(y/n).

Was[No], Now is:[Yes].

Displaying the Configuration File

To display the contents of the configuration file use the **conftable** command. **conftable** reads the information from a file or a group of files and displays the information in a list to standard out. For example, to display the information in the file created above with the **confedit** command, type

ADMIN>conftable conf

The display appears as

Configuration from file 'conf'

*/ROOT device is 4, 10.

Swap device is '/dev/swap'.

Swap area is 8000 blocks.

80 system buffers allocated.

100 process control blocks allocated.

Ceiling priority is 100 for start process.

Floor priority is 32 for start process.

Swapper priority is 200.

Maximum logical units is 400.

Maximum number of open files is 200.

Maximum number of event calls is 100.

Maximum number of shared data segments is 200.

Priority tuning interaction credit is 512.

Priority tuning disk wait credit is 0.

Priority tuning quantum used up demerit is 8.

Input buffers are 256 bytes per tty.

Maximum mounted file systems is 8.

Histogram log area is 2000 bytes.

Clock chip rate is 1250000 Hertz.

Will not wait before mounting ROOT.

Maximum number of locks (lock lists) is 40.

Default erase character is '\177'.

Default line kill character is '\25'.
System name is ''.
5 blocks in multi block I/O buffer.
1024 bytes in system error log buffer.
Recovery ROOT device is 4, 8.
Script is '/etc/startup'.
Auto start.
Mount ROOT read/write.
Number of pseudo teletype devices is 2.
Poll after 100 services.
Distributed systems structures memory is 0.
Number of ipc queues is 0.
Total number of ipc message bytes is 0.
Maximum size of a single ipc message is 0.
Maximum ipc message bytes on one queue is 0.
All created files are not contiguous.
Non contiguous file allocation chunk size is 0 blocks.
Default minimum contiguous chunk size is 0 blocks.
Default maximum contiguous chunk size is 0 blocks.
Outgoing distributed file system requests are enabled.

End of Configuration 'conf'

To find the current configuration on your system, type
ADMIN>conftable -c

The configuration file shown in *The Configuration Table* is displayed.

If you name a file that is an illegal configuration file, the message
***** NOT A CONFIGURATION FILE ***** is displayed.

confedit returns these error messages

Format error.

input makes no sense in the current context. Before the error message, **Unchanged** is displayed.

Syntax error.

a parameter name is not recognizable, issued in non-interactive mode .

Non unique abbreviation.

parameter name is not unique, issued in non-interactive mode.
No checking is done on pathname strings for illegal characters.

For more information about the **confedit** and the **conftable** commands, see the *UNOS Commands Manual*.

Example

Some systems can improve their I/O performance by changing the number of system buffers in their configuration file. Use these guidelines to determine if changing the system buffers will enhance your I/O performance.

- a system with 2 MBytes or more memory
- same files accessed repeatedly
- same command used repeatedly

System buffers allow more information to remain in memory rather than on the disk, making access faster. You must weigh faster access against reduced memory and increased buffer searching time. For example, a 2 MByte system configured with 60 system buffers contains 1780 KBytes of memory after booting. By changing the number of system buffers to 300, the system has 1648 KBytes of memory left after booting. Another consideration is time. Will it be more time consuming to search 300 buffers for your file or go to the disk?

Systems that run application programs can benefit from more system buffers if the same files or commands repeat. If the application program accesses and writes new information constantly, the system is not a good candidate for enlarging system buffers.

Chapter 3

UNOS Priorities

Since every process on UNOS has an associated priority, a ready process with higher priority will always run before one with lower priority. For time-critical tasks you can set up, a fixed priority structure to guarantee the desired precedence essential in a real-time system.

Priorities can also be used to ensure good interactive response time, critical in a time-sharing system. UNOS, normally configured for time-sharing, assigns higher priority to processes that involve user interaction, such as the editor (**ted**) and commands that answer questions, such as who is on the system (**who**). While CPU intensive commands, such as C program compiles (**cc**) and searching (**match**), get lower priority. The system continually adjusts processes so they remain at a reasonable priority. A high priority alone does not guarantee that a process will not get swapped to disk. The swapper simply looks at the quantum of a process, and once it is delpteted will swap it out regardless of the process's priority.

An additional mechanism for controlling resource allocation is *process locking*. A locked process is not swapped out of memory until it has completed. Consequently, no time is lost reloading the process from the swap device. In situations where a process must respond very quickly to input, you would not want to waste time fetching it from the swap device. To really ensure that the process is always ready to run, however, you must set its priority high and lock it in memory.

Although process locking is a very powerful tool, use it with discretion, since it affects the entire system. Before locking a process, make sure adequate memory is available, since locking fragments memory and reduces the size of the largest process that can run. Also bear in mind that locking processes shrinks the total amount of user space on the system. For less fragmentation, start the locked process in low memory. To do this edit the **etc/startup** command

file and place the locked process's command line as the first item in the file.

NOTE: Do not lock daemon processes.

The Way Priority Works on UNOS

The priority values can range from 0 to 255. The distributed configuration file, */etc/conf*, sets the limits on user processes of 0 to 100 and sets the swapper's priority to 200. You can set administrative processes higher than user processes or the swapper, but do this with caution.

Your shell starts at or near a ceiling of 100. When it forks to the child process, the priority and quantum (limit on CPU time) are inherited. Whenever the process uses up its quantum, its priority is lowered (by the amount specified in the *quantum used up* field in the configuration file) and its quantum increased. Its priority, however, is raised and its quantum reduced when it waits for terminal input/output, pipes, children or eventcounts. The amount it is increased is proportional to the value in the *interaction credit* field in the configuration file. The system makes an attempt to balance the priority with the amount of time it takes to complete the job.

The system administrator can raise the priority of processes so that they run before the time-sharing processes. The administrator can also avoid fluctuations in priority by setting a process's floor and ceiling to the same number. If several processes are set at the same priority, they will be run in a round-robin fashion.

An Example - *ted*

ted begins with a priority of 100, but then after it has read in the terminal descriptor files, the editor program and the file to edit, its priority drops to about 80. Every time it waits to read in a character though, its priority climbs. This is because *ted* never computes long enough to deplete its quantum, and therefore, lower its priority. Therefore, after the initial read-in, *ted* runs at nearly 100 for most of the editing session.

Priority Commands and Subroutines

The UNOS commands for priority control fall into two categories: setting and displaying priorities and locking processes. Users can lower their processes' inherited priority limits, but only the system administrator can raise priority or lock processes.

Setting Priority

The **nice** command executes a process and sets its priority limits and or locks it in memory.

NOTE: You can set priority and locking independently of each other.

The command syntax is

```
nice [ceil=#] [floor=#] [-lock] command line
```

Suppose we wanted to merge and sort large files like dictionaries. This is a CPU intensive command that impacts system performance. We could run it at a lower priority, and thus, not slow down the system for the other users. The following command line runs the **sort** program at a lowered priority:

```
@>nice ceil=40 floor=10 sort o=new.dict new.list dictionary
```

While the **nice** command executes a command at a specified priority, the **setpri** command alters the priority of a running process. If you do not specify a process ID, it displays the priority of the current process. This command helps the system administrator manage the system.

The command syntax is

```
setpri [ceil=#][floor=#] [pid=#]
```

In the following example, we lower the priority of a large formatting command so that it does not infringe on interactive processes.

```
ADMIN>setpri ceil=50 floor=0 pid=235
```

The **notify** command is similar to the **nice** command, differing in that it backgrounds the process, has optional output files and notifies you when the command completes.

NOTE: Using **notify** or **nice** without stipulating priority, lowers the ceiling priority by a fixed amount (20). Also, if you background a process by using an ampersand (&), its priority will be lowered by 20.

Displaying Priority

The **ps** command displays the process status of your processes which includes its current priorities. If you use the **-all** flag, all the processes on the system are displayed. The **-long** flag displays the complete information about processes. In the following example, we have requested the long process status,

```
@>ps -l
```

```
PID USER FLG STAT PRI MAX MIN Kb SYS USER CHILD TIMES AGE LU INT SIGS
415 szy  820 WAIT  94 100  32 56 0.7  0.1  0.8  0.1 3.7  5  0  0
WAITFOR 30B2C child
PROGRAM -
```

```
577 szy  0  RUN   65 100  32 40 0.6  0.1  0.0  0.0  0.3  8  0  0
PROGRAM ps -l
```

The **-long** flag displays ceiling (MAX) and floor (MIN) priorities. It displays the time usage of the child process and the age (AGE), which is the time the process has spent on the current queue. It tells you the number of logical units (LU's) the process is using; the maximum is 100 logical units per process, and it gives you the interrupt and signal bits for the processes.

Priority Subroutines

On the programming level, there are four subroutines that deal with priorities: **getpriority()**, **setpriority()**, **lock()** and **nice()**. The first three function like the corresponding commands. The **nice()** subroutine is mutually exclusive of the first three. It is

UNIX-compatible and uses the same interface; therefore, the numbers correspond to the UNOS range. **nice()** alters the ceiling value. For more information about these commands, refer to the *UNOS Subroutines Manual*.

Tuning Priorities

On the daily level, the system administrator can use the **ps** and **setpri** commands to monitor and adjust the priorities on the system. For more lasting modifications, the system administrator can make changes to the **/etc/conf.c** file. This file controls timing parameters: initial priority ceiling and floor; the priority credit for an interaction or for disk I/O; the size of the decrement for using up the quantum; and the relative importance of priority and time-on-queue in determining which processes should be swapped out. Refer to the *UNOS File System* chapter for more information about the **/etc/conf.c** file.

Chapter 4

Backup and Restore Procedures

This chapter explains how to back up and restore files directories and file systems. Use the **debe**, **package** and **installation** commands for back ups and restorations.

The *Back Up and Restore Procedure* chapter describes

- full and incremental back ups
- administrator functions to facilitate back ups
- automatic back up scripts
- back-up media library and logs
- file system restoration using the **debe** command
- file restoration using the **installation** commands
- examples of full and incremental back ups and restorations to floppy disks and 1/4" tape

Back Up

The process of periodically dumping the file system is called *file system back up*. Periodic file system dumping maintains copies of users' files. Use the **debe** and **package** commands to back up. **debe** does a block-by-block copy of a file system, whereas **package** backs up specified files.

Dumping the file system involves copying all or part of a file system's contents. Although you can copy the file system onto a different part of the disk, for safety reasons, dump the file system to a removable medium, such as a floppy disk or magnetic tape.

The **package** and **debe** commands allow you to do a file system back up. You can do an incremental back up with **package**.

A full back up dumps the contents of the entire file system. An incremental system back up saves only the files that you specify, regardless of whether the last back up was full or partial.

Media is an important consideration when backing up. 1989 blocks is the maximum allowed on a floppy disk. For larger volumes you would have to replace the floppy disks every 1989 blocks when the **package** command prompted you. You can back up 40 Mbytes of information on a 1/4" tape.

package

package writes a tape or floppy in UNIX cpio format, which the UNOS **installation** program reads. **package** reads the files from the command line and recursively dumps the files to standard output. Recursively dumping files means **package** dumps subfiles and directories of specified files on a partition, like **/usr**. Make sure the partition is mounted. Use **package** to dump an entire system at once. If no files are specified on the command line, **package** reads the standard input for the file names to dump. Use this command to set up *take* files with the **list** command. If **package** comes to the end of file on an output medium, it requests a new volume to begin writing to.

package also

- recovers files from filesystems with bad blocks
- reports any unreadable blocks
- backs up entire running systems by file modification date
- *streams* a 1/4-inch tape by using the **bs=** option to specify the amount of buffer memory to use.

The **package** format is

```
package [bs=buffer size] [after=date] [-s] [-q] [-c]
[ib=MMddHHmmYY] [-recover] [-help] [-nr] [-x] [-v] [-tape]
[patterns]
```

The **package** command options include

bs=buffer size denotes buffer size. The default buffer size is 5120 bytes. Specify the buffer size with *b* for block (512 bytes) or *k* for kbytes of memory to use.

after=date dumps only the files appearing after the specified date.

Use option for incremental back ups of a system. Specify the date in the same string format as the UNOS **setdate** command, **MMddHHmmYY**.

MM Month in numeral format. (01 is Jan, 02 is Feb, etc)

dd Day of month.

HH Hour in 24 hour format. (9:00 PM is 2100)

mm Minutes.

YY Last two digits of the year. (1985 is 85)

-s indicates silent mode.

-help specifies the standard UNOS usage function for the on-line description.

-q prompts the console with the file name and asks for a response of y or n. An n response does not save the file.

-c adds up all the sizes of the files that would be dumped if this option was unspecified.

-recover allows the program to continue from disk read errors on files and directories. If the block is unreadable,

- the area of the buffer that would have contained the block is NULLED (zeros written)
- the file name and logical block in the file is reported to the console
- the program continues with the file

If the error is on a directory

- the directory block is skipped
- the directory name is reported with a warning that some entries may not be saved

Use this option to save files from a disk with bad blocks.

-x specifies the first name on the command line as the device to write the output to.

-nr does not dump any subfiles or directories of any named files, non-recursive.

ib=MMddHHmmYY (incremental back up) provides for incremental back ups of any files with a date/time that falls between the current time and the current time minus the option argument string.

-tape closes the device between volumes so that the device driver can write a file mark on the tape, and not latch the "tape changed" status. You must specify the **-x** option also so the program knows the name of the device to open for the next volume.

debe

debe copies the specified input to the specified output performing the requested conversions. The standard input and standard output are used by default.

debe [option=value] [-flag] ...

The **debe** options include

-fill pads every output record with zeroes to the requested size.

-pg prints a dot each time a record is written to indicate progress.

iskip=# skips # bytes of the input file before beginning transfer.

oskip=# skips # bytes of the output file before beginning transfer.

skip=# combines the functions of **iskip=#** and **oskip=#**.

ivskip=# skips # bytes at the beginning of each input volume. The **iskip=#** option still works, but only applies to the first volume, where it adds its value to the value of **ivskip**.

ovskip=# skips # bytes at the beginning of each output volume. The **oskip=#** option still works, but only applies to the first volume, where it adds its value to the value of **ovskip**.

ivsize=# specifies the maximum length, in bytes, of each input volume. When the end of the volume is reached, **debe** asks for the next volume to be mounted. It waits for you to type a Y <cr> before beginning the new volume. An N response is treated as an end of file condition and **debe** writes any buffered data to output and exits.

ovsize=# specifies the maximum length, in bytes, of each output volume. At the end of each volume **debe** prints a message

requesting that you mount the next volume. It waits for a Y <cr> response before it begins the new volume. An N response stops the process and **debe** prints statistics that show more data was read than written.

ibs=# specifies input block/buffer size in bytes.

obs=# specifies output block/buffer size in bytes.

bs=# specifies both in/output block size in bytes.

if=name specifies input file name.

of=name specifies output file name.

-noerror continues transfer despite errors. Error messages will appear on the screen.

count=# transfers # of input records or until EOF.

-notrunc does not truncate an already existing output file, enables copying one file into another.

-swab swaps bytes (except for the last byte in odd block sizes and odd transfers due to EOF).

Considerations When Backing Up to Tape

On systems with 1/4" cartridge tape, you can use tape to back up your system. This section includes the following information:

- recommended back up techniques
- tape partitions
- skipping blocks when copying
- tape seeks
- buffer sizes

You'll find several specific examples in the *Examples* section at the end of this chapter on how to back up and restore, how to copy particular files from a tape, and how to do incremental back ups .

Recommended Back-up Techniques

Keep the following points in mind, to simplify your back ups.

1. Back up complete file systems on a regular schedule.
2. Use the largest possible buffer size while reading and writing from the tape to make the operation go faster.

NOTE: The buffer size must be a multiple of 512.

3. Never use the buffered tape device (`/dev/ct0`, `/dev/ct1`, etc.); use the "raw" type devices (`/dev/rct0` ...). Buffered devices usually do physical I/O in 512-byte blocks.

Tape Partitions

The 1/4" tape partitions are dynamic. A tape partition is the area either between the beginning of tape and a tape mark or between two tape marks. Tape marks are written whenever the tape device is closed or when the tape is rewound after writing at least one block.

The last written tape mark defines the end of tape. You cannot read beyond it because either

1. No information exists (i.e., when it's the first time a tape is being used);

or
2. You can only read from the beginning of the tape. Any break (i.e., old data from a previous write) destroys the synchronization.

However, you can begin writing again just past any tape mark. The hardware first reads the tape before the tape mark and begins to write synchronously just after the tape mark, creating a new partition. This means that when you write, you always write to the last tape partition. For more information about cartridge tape, refer to the *Device Organization* chapter in this manual.

Because of the way you write to tape, you cannot write to a partition without *erasing* all partitions after the one you write to. For example, if partition 0 was 20,000 bytes long and partition 1, 2, and 3 existed, even if you only wrote 1000 bytes to partition 0, its newly written tape mark would define the end of the tape, and partitions 1, 2, and 3 would be gone along with their data.

Skipping Blocks When Copying

You cannot do **fileseek()** while writing to a tape — the tape cannot count bytes before they are written. If you use **debe** to write to the tape, you cannot use the *oskip* or *skip* options.

Tape Seeks

Every time a read requests a block that is located physically before the current tape position, the tape rewinds to the *beginning of tape* and begins counting blocks until the tape reaches the requested block. Therefore, if you use a tape as a random access device (a mounted UNOS file system) you will incur many rewinds. Repeated tape rewinds is time-consuming, especially if you are not using partition 0.

NOTE: Tape blocks are 512 bytes.

Buffer Sizes

The larger the read or write request, the less often the tape needs to stop. When the tape restarts, it needs to back up a bit to get up to speed before it can begin to read or write again. With small buffers, this stopping and rewinding can add up to a considerable amount of time.

New Tapes Initialization

New tapes should be initialized prior to use. To do this, type
\$iostats -erase -rewind /dev/ct0

General Back Up Procedures

You should do incremental back ups every day, and a full file system back up as you deem necessary. You might want to back up an active file system weekly, whereas a less active file system may get backed up monthly. Do back ups either in the morning--right after booting the system-- or at the end of the day, either time when the system is running in single user mode. Therefore, the person booting the system in the morning, or the person shutting down the system at night, is responsible for the incremental back up. The system administrator should do the full system back up.

These instructions explain the back-up procedure. For specific back-up examples, see the *Examples* section of this chapter.

1. At the system console bring the system to single user mode. To do this, log in as the system administrator; that is, type **root** or **admin** to the **login** command. Or, type the **setuser** command without any arguments. (What you type depends on how you configured your system.) Then shutdown the system by typing **shutdown**. For more information about these two commands, refer to the *Starting Up and Shutting Down UNOS* chapter of your *Universe System Installation Manual*.

2. For a back up onto tape or floppy disk, type
#package /mounted-file-system > /dev/to <cr>
or
#debe if=/dev/from of=/dev/to bs=xb ovsize=xb <cr>

debe and **package** prompt you for more floppy disks, if necessary.

For an incremental back up onto tape or floppy disk, type
#package after=MMddhhmmyy files > /dev/to <cr>

3. Enter the correct information in the back up log.

If a problem develops, you should rerun the entire back up procedure.

Administrator Functions For Back Ups

Create administrator functions for full and incremental back ups. Functions save time and prevent errors. Make sure the functions reside in administrator's *.profile* file.

The following are sample functions for a full and incremental back up of a Charles River Universe System. Use the sample functions explanations as models to create functions for your system. If you need more information about how to create functions, refer to the *User's Guide to UNOS*.

Functions have this form

```
function functionname <cr>
{<cr>
shell script <cr>
}
```

To create a function, you must use your editor to edit your *.profile* file. Add new functions to the *.profile* file. Leave the editor, saving the new functions. An example of a full back up function is

```
function fullbackup
{
package / bs=198b > /dev/rcfd0;
echo "Full back up completed at:" >> /log/backup_log;
/bin/date >> /log/backup_log
}
```

The function fields mean

- function identifies this as a multiple command sequence or a general command
- fullbackup is the name of the function
- package is the name of the command that initiates the back up
- / indicates the pathname to where the backed-up files will reside
- /dev/rcfd0 indicates the floppy where the files will be copied to (destination device)
- ; specifies the end of the **package** command sequence
- echo initiates the second command sequence and appends the

-
- statement to the file `/log/backup_log`
 - `;` specifies the end of the **echo** command sequence
 - `date` initiates the third command; **date** executes and appends the date and time to the file `/log/backup_log`

For instance, if you execute this function the following entry is saved in the file `/log/backup_log`.

Full backup completed at:
Tue Aug 26 1986 17:14:34

The following example backs up the files changed within the last twenty-four hours on the `/usr` partition.

```
function incbackup  
{/cmd/find /usr time=1 | package -nr > /dev/rct2}
```

The function fields mean

- function identifies this entry as a multiple command sequence or a general command
- `incbackup` is the name of the function
- `/cmd/find` is the UNOS command to find files altered within 24 hours
- `/usr` the directory `/cmd/file` looks in for the altered files
- `time=1` time parameter for changed files -- one day, 24 hours
- `|` pipe that feeds the information to the next command sequence
- `package` is the name of the command
- `-nr` option that indicates no recursion
- `>` indicates where to put the information
- `/dev/rct2` indicates the device where the files are (source device)

Automatic Back Up Script

You can put your back up in a `/lib/chrontab` script, then your system will back up automatically every day at a specified time. However, this automatic back up works best for tape unless you are backing up less than 1989 blocks. For example, the `/lib/chrontab` script below backs up the `/usr` files specified in the script `/backup_user` on the system called `crypt` every night

```

#/lib/chrontab
#
#Mins      = 0-59 range,
#*         = any time matching other fields
#Hours     = 0-23   range
#DWeek     = 0-7   1 is Monday, 0 or 7 is Sunday
#Month     = 1-12   range
#MDay      = 1-31 range
#Command   = file or program to be executed
#Mins      Hours      DWeek      Month      MDay      Command Line
#
# run nightly backup of the user files
0          0          1-7        *          *          /backup_user

```

The lines that begin with # are comment lines. The **chron** command executes the file, *backup_user* every night at midnight. */backup_user* includes all the */usr* directories on *crypt* that change daily. **chron** sends a message to */log/backup_log* when the file completes the back up.

You can create a similar file that **chron** can execute. An example of */backup_user* is

```

package /usr/charlie /usr/cjk /usr/dottie /usr/dianne
/usr/ewheeler /usr/ken /usr/kirpal /usr/maureen /usr/informix
/usr/sandra /usr/ted bs=1024k > /dev/rct0;
iostats -rewind /dev/rct0;
echo "User back up completed at" >> /log/backup_log;
/bin/date >> /log/backup_log

```

Back Up Media Library and Log

Because maintaining back-up media is important, we want to describe a back-up media library and log program. You can use the program we outline or create your own.

Maintain a library of the back-up media. This library should have two sets for full back ups (labeled *your_system full set 1* and *your_system full set 2*) and two sets for incremental back ups (labeled *your_system inc set 1* and *your_system inc set 2*).

The incremental sets are used with the same numbered full set; i.e., when you use the *your_system full set 1*, the successive daily dumps are done on *your_system inc set 1*. Alternate the sets.

Keep a written log of back-up activities. After each back up, make an entry in this log. A complete entry consists of

- the date of the back up
- the type of back up (full or incremental)
- the disk set used (1 or 2)
- the number of disks used
- the day/vol numbers of the disks used for an incremental back up
- the name of the person who performed the back up

Restore

The **installation** and **debe** commands restore files. Use **installation** to restore files backed up with **package**; use **debe** to restore file systems **debe** backed up.

installation

installation restores files the **package** command writes to media. command. You can restore the entire file system or selected files from the dump with **installation**. The back up volumes are read from *input_device*.

The command syntax is

```
installation [-xvbtbf] [-recover] [bs=#] [pattern]
```

The options to **installation** include

- x lets you name a device to read from, rather than reading from the standard input.
- V indicates multi-volume sets for an archive that includes many tapes or disks, this option expects information at the beginning of each one that instantly indicates misordered volumes.

NOTE: This option is NOT *cpio* compatible.

-
- b tells the **installation** command to expect a 512-byte block.
 - t extracts no files, but instead prints those files to standard out.
 - c tries to install all files as single extent contiguous files. If the -c option can not install the files as contiguous, creates the files as a standard block file.
 - s *string* strips off the front of a pathname string and installs the string into the pathname of the current working directory.
 - recover recovers from abnormal file dumps. Use this option to recover from a bad back up or to recover a lost volume, for example, the third floppy of 10.
- bs=#* reads from 1/4" tapes to allow the tape to stream. The memory is rounded to the nearest 5120-byte block for *cpio* compatibility. You can specify # in bytes, #k for 1024-byte chunks, or #b for blocks or 512-byte chunks, for example, *bs=100k*, *bs=100b*, *bs=25600*.
- f does not copy the patterns you specify.

Upon completion **installation** displays a message similar to:
100666 8, 0 1662 bytes Thu Aug 3 11:56:54 1986 /src/ter

The message means

- 100 is the directory or the file
- 666 is the permissions on the file
- 8 indicates the user id
- 0 indicates the group id
- 1662 bytes is the size of the file
- Thu Aug 3 11:56:54 1986 when the file was written to tape
- /src/ter is the file's name

For additional information about the **installation** command, refer to the *UNOS Commands Manual*.

General Restore Procedures

Make sure you are admin.

If you backed up with **package**, use the **installation** command to restore. If you used a relative pathname, change to the appropriate directory.

```
#installation < /dev/xxx0
```

Use **debe** to restore the same way as you would to back up. See the *File System Back Up* section for an explanation of **debe** to back up a file system.

Examples

The *Examples* section gives specific back up and restore situations. These examples include

- Back Up and Restore the First Disk Partition
- Back Up and Restore a Random Disk Partition
- Store and Copy Tape Files
- Back Up and Restore Selected Files
- Back Up and Restore Directories

Back Up and Restore the First Disk Partition

To back up and restore the back-up root of a Charles River Universe system, you need to do the following:

For back up:

1. Log in as admin
2. Make sure that **/dev/rck0** is unmounted.
3. Copy the disk partition to the tape.

Using the **debe** command:

```
#debe if=/dev/rck0 of=/dev/rct0 bs=500b
debe: Read 4 records (total of 1024000 bytes).
debe: Wrote 4 records (total of 1024000 bytes).
```

To restore:

1. Copy the tape partition to the disk, skipping the first six blocks on both the tape and disk.

```
#debe if=/dev/rct0 of=/dev/rck0 bs=500b
debe: Read 4 records (total of 1024000 bytes).
debe: Wrote 4 records (total of 1024000 bytes).
```

NOTE: The first 9 blocks do not need to be skipped, because this is not the first physical disk partition -- there is no bad block map or bootstrap information to worry about.

Back Up and Restore a Random Disk Partition

The following example describes how to back up and restore the ck2 partition from a Charles River Universe system:

1. Make sure that the disk partition is unmounted.
2. Find out how much of the disk has been used:

```
#space /dev/rck2
/dev/rck2: 59029 free out of 72000 blocks (high 12972)
```

3. Copy the used (up to "high") part of the partition to the second tape partition.

```
#debe if=/dev/rck2 of=/dev/rct1 bs=500b count=26
debe: Read 26 records + 0 bytes (total of 6656000 bytes).
debe: Wrote 26 records + 0 bytes (total of 6656000 bytes).
This copies 13000 blocks (26 x 500), 28 more than used on
partition /dev/rck2.
```

4. Tape partition 1 is now full at 13000 blocks. To write more to the tape, we will use the next partition, ct2.
5. Restore the disk partition with the following command

```
#debe if=/dev/rct1 of=/dev/rck2 bs=500b
debe: Read 26 records + 0 bytes (total of 6656000 bytes).
debe: Wrote 26 records + 0 bytes (total of 6656000 bytes).
(See the NOTE on page 125)
```

Store and Copy Tape Files

To save all the files belonging to **/usr/bob**, type

For floppy:

```
#package /usr/bob/* > /usr/rcfd0
```

For tape:

```
#package /usr/bob/* > /usr/rct0
```

These **package** command lines write all the files in **/usr/bob** to a specified absolute pathname, **usr/bob/filename**. If you want the files installed in a directory other than **/usr/bob**, change the directory

```
#cd /directory
```

and execute the command line

```
#package * > /dev/output_device
```

You can mount a floppy disk and copy files to it. For a new floppy, you must format and make a file system on the disk. Use the **fmtdisk** and **makefs** commands.[1] Follow these directions to mount a new floppy and copy all the **/usr/bob** files onto the floppy.

1. Format the disk

```
#fmtdisk -force /dev/rcfd0<cr>
```
2. Create a file system

```
#makefs /dev/rcfd0 1989<cr>
```
3. Mount the disk

```
#mount /dev/rcfd0 /mnt<cr>
```
4. Copy files

```
#copy /usr/bob/* /mnt<cr>
```

1. See the *UNOS Commands Manual* for more information about these commands.

-
5. Unmount the disk
#unmount /dev/rcfd0 <cr>
or
#unmount /mnt <cr>

As an alternative method of saving specific files place specified files into a *takelist* file and use the *takelist* file in a **package** command line.

For instance, the *takelist* file looks similar to:

```
/usr/sandra/.aliases  
/usr/charlie/.profile  
/etc/termcap
```

To write the three files in *takelist* out to a floppy in **installation** format, type

```
#installation -nr < /takelist > /dev/rcfd0 bs=198b<cr>
```

Back Up and Restore Selected Files

This example backs up all the files that match the pattern **ter** beginning with the present working directory.

```
#package *ter* > /dev/rmt0
```

This **installation** example installs any files from the tape */dev/rmt0* that match the pattern **ter** from the */* directory.

```
#installation -x /dev/rmt0 '*ter*'
```

```
100606 88, 0 18625 bytes Mon Feb 13 18:48:49 1984 /src/ter/T  
100606 18, 0 451 bytes Fri Aug 10 09:53:59 1984 /src/ter/M  
100606 101, 0 3079 bytes Wed Jan 4 10:20:30 1984 /src/ter/N  
100666 88, 0 17817 bytes Wed May 9 12:15:39 1984 /src/ter/A  
100666 88, 0 16626 bytes Thu Aug 23 11:56:54 1984 /src/ter/H
```

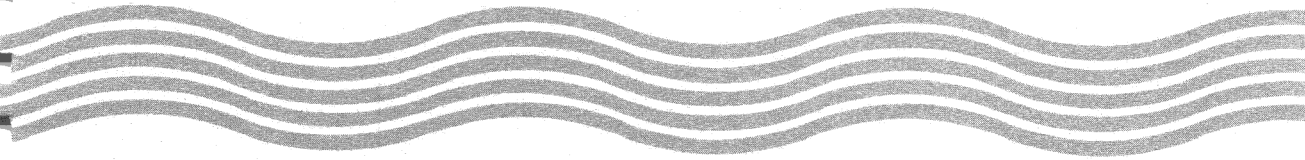
Back Up and Restore Directories

To back up the root using the **package** command, type
`#package / > /dev/rcfd0`

To restore the root, type
`#installation < /dev/rcfd0`

To back up just the root, you must unmount the other file systems.

Section 3
Reference



Contents

131	Glossary
139	Device Node Table
147	Index

Glossary

This section defines basic terms. The terms appear in alphabetical order rather than in order of usage. Read it before you use your system. For more information about using UNOS, refer to the *User's Guide to UNOS* and the *UNOS Commands Manual*.

admin

The system administrator is the privileged user. You can become the administrator by using the login name **admin** and knowing the correct password. When you login with this name, you have a different system prompt. Instead of the default prompt, **@>**, you have the administrator's prompt, **ADMIN>**. If your default shell is the Korn Shell, your default prompt is **?** and the administrator's prompt is **#**.

Backup Root

The backup root is an alternate bootable file system root. You can use it when there is an error on your main root. To fix your main root, you run off your backup root and then reboot the system. Refer to the *Automatic Power Up and File System Checking* chapter of this manual for an explanation of the main and backup roots.

Bad Block Map

A listing of unusable areas on a disk.

Bit Map

An effective means of distinguishing empty from busy blocks on a device.

Block

512-byte section of a device.

Boot Device

The boot device is the logical device from which the operating system kernel, IOP and configuration files are read. It may or may not be the same as the root device. Refer to the *UNOS File System* chapter of this manual for more information about the device configuration.

Bootstrap Routine

This program boots the operating system.

Buffered Mode

Buffered mode means the data is transferred in 512-byte units.

Command Display

Throughout this manual we show what the system displays in **typewriter bold type face** and what you, as a user, should type in typewriter typeface.

Command Syntax

You type all commands after the system prompt and separate all command arguments with spaces (spaces are inserted by pressing the space bar) or tabs (tabs are inserted by pressing the TAB key). There are two types of command arguments: modifiers and filenames. Modifiers are preceded with dashes or have embedded equal signs, everything else is usually considered a filename.

Simple modifier arguments are typed in the following way:

```
@>list -long -all
```

This command prints a long listing of all the files in the directory.

Modifier arguments that have values are typed in the following way:

```
@>lpr printer=fancy chapter1
```

This command prints the file, *chapter1*, on the fancy NEC sprinwriter printer.

Each command is terminated with a carriage return; this is displayed on sample dialogues as <cr> (which does not actually appear on your screen). When we display commands and all their arguments, the optional arguments are indicated by square brackets, for example, **list [-a] [-long]**, means that you could type the **list** command alone or with either one or both arguments. (See the earlier use of the **list** command.)

Console

The console is the terminal attached to port 0. It receives all the system error messages and is the only terminal that can startup and shutdown UNOS.

Contiguous files

Contiguous files consist of contiguous areas of disk called extents.

Control Characters

Control characters are single, nonprinting characters that affect commands and terminal input and output. They are typed by holding down the CTRL key and pressing a character key at the same time. They are displayed as two characters on sample dialogues with the notation **^X** where **^** is the control character and **X**, is the character. Two frequently used control keys are Control C and Control D. Control C, **^C**, kills the current command. Control D, **^D**, when typed in the middle of a line, means end-of-line and displays the message **EOL**. This forces the line into the system without a trailing new-line character. When **^D** is typed at the beginning of a line, it means end-of-file and displays the message **EOF**, sending an end-of-file to your Command Interpreter which will then log you out. See Appendix F in your respective *Universe System Installation Manual*.

Default

Defaults are the system choices when you don't specify a choice.

Device Node

A device node is a mapping of a device (logical or physical) to a file name. By convention, all device nodes are located in the `/dev` directory. They all have a corresponding Major/minor number, abbreviated as MM/mmm. The Major number refers to the particular controller, while the minor number can mean various things. In the case of disks, it refers to the logical subdivision on the device. For more information about device nodes, refer to the *Device Organization* chapter in this manual.

Device Switch Table

A device switch table is an index of the system's drivers.

Driver

A driver is software that controls a peripheral device.

Extents

Extents are contiguous chunks which can be from one block to two giga blocks long.

File Descriptor

A file descriptor contains information about the file it describes, such as the file type -- data, device, or eventcount, permission modes on the file, the number of links and various control bits. Every file the system might have has a file descriptor. Unless otherwise noted, the system allocates 9/64 the number of blocks for file descriptors -- this is roughly one file descriptor for every eight disk blocks.

File System

A file system is a complete directory structure (including the root and its subdirectories) on a logical device. A file system exists on a physical device, which is associated with some form of magnetic media, such as a hard disk, a floppy disk or tape. A file system can be any size. UNOS can only access a file system after it has been mounted. For more information about file systems, refer to *UNOS File System* chapter in this manual.

Floppy

A floppy disk is a removable disk, either 8" or 5 1/4" in diameter.

Floppy Image

A floppy image is a floppy-sized piece of a device.

Input File

An input file is data that is entered into the system.

Interleave Factor

The interleave factor is the number of sectors that pass between two consecutive read or write commands to allow the disk controller firmware to set up for a new command.

Logical Device

The system treats each logical device as a separate unit of information storage. Large capacity devices can be further subdivided into smaller logical devices, each of which can contain a complete file system. This partitioning is controlled by a partition map stored in the PVH that resides on the space provided for it. For more information about physical and logical devices, refer to the *Device Organization* chapter of this manual.

Magic Word

A magic word is a value.

Major Device Number

A major device number designates a certain driver for a device. The abbreviation for a major device number is MM.

Minor Device Number

A minor device number designates which logical device a driver is addressing.

Modem

is a device that allows you access to your system over telephone circuits. The name is an acronym of Modulation/Demodulation device.

Mount

To mount a device or file system gives the user access to that device or file system.

Named Pipe

A named pipe is a single file descriptor, plus the required data blocks.

Operating System

An operating system provides the user with orderly access to the hardware and its resources.

Output File

A file that includes data from the system.

Partition Record

The partition record is located in the first block of the physical device. It contains information about how the disk is subdivided into logical devices. For more information about partition records, refer to the *Device Organization* chapter of this manual

Pathname

The pathname uniquely specifies the logical *path* you would take to locate a file or a directory in the file system. You begin at the root (represented with the symbol /) and name the sequence of directories that come before the file or directory you want. Each directory is separated with a slash.

Peripheral

A peripheral is a hardware device, for example, a disk or a terminal.

Physical Device

A peripheral is a hardware device, for example, a disk or a terminal.

PVH

A PVH is a physical volume header. Through the PVH, a disk driver determines the characteristics of a disk.

Root Device

The root device refers to a logical device selected at boot time by the configuration file to be the top-most level of the file system. For more information about the configuration file refer, to the *UNOS File System* chapter of this manual.

Setuser

The **setuser** command lets you overlay your initial login; that is, after you have logged in as yourself, you can type this command and become another user. If you type the command without a user name, you become the system administrator (you must also know the correct password). If you type the command with a valid user name (and know the password), you become that user. **setuser** saves time, because it lets you change your identity without logging out. For more information, see the **setuser** command page in the *UNOS Commands Manual*.

Stepper Pulse

The signal to the heads to move to the next track.

Swap Pathname

The swap pathname specifies the logical device assigned as the system swap device. The swap device is not a standard file system; it is an extension of memory. When the system memory is full, the system temporarily saves a waiting process on the swap device in order to make room for processes that need to run. For more information about this device, refer to the *UNOS File System* chapter in this manual.

tpi

tpi means tracks per inch on a floppy disk.

tty

A **tty** is a physical port.

Unbug

Unbug is a ROM that includes three program: diagnostic self-test, bootstrap to load the operating system, and debugging firmware.

Unboot

Unboot is a program that runs the operating system from the disk.

Unbuffered Mode

See **buffered mode**

Virtual Floppy

Storage space on a disk equal to the size of a floppy disk.

UNOS Device Table

The following table contains section names and block sizes for all the standard devices on UNOS.

M	m	Name	Use	# Of Blocks
0	0	/dev/null	Black hole	
0	1	/dev/kmem	Kernel memory	
0	3	/dev/tty	Controlling terminal	
1	0	/dev/tty0	serial port 0 (console)	
1	1	/dev/tty1	serial port 1	
1	2	/dev/tty2	serial port 2	
1	3	/dev/tty3	serial port 3	
1	4-67	/dev/tty4-67	(TP-304 and TP-308 only)	
1	128-134	/dev/par0-6	parallel ports (on TP-304 and TP-308 only)	
ca				
4	0	/dev/ca0	Backup root	1988
4	1	/dev/swap	Swap	2000
4	2	/dev/ca2	Root	36000
4	3	/dev/ca3	Entire disk	40000
cb				
4	0	/dev/cb0	Backup root	1988
4	1	/dev/swap	Swap	2000
4	2	/dev/cb2	Main file system	16400
4	3	/dev/cb3	Entire disk	20808

M	m	Name	Use	# Of Blocks
		cd		
4	0	/dev/cd0	Backup root	1988
4	1	/dev/swap	Swapping unit	2000
4	2	/dev/cd2	Main file system	20000
4	3	/dev/cd3	User section	20000
4	4	/dev/cd4	Spare section	20000
4	5	/dev/cd5	Spare section	40000
4	6	/dev/cd6	(cd3,cd4)	60000
4	7	/dev/cd7	Entire disk	65536
		cj		
4	0	/dev/cj0	Backup root	1988
4	1	/dev/swap	Swapping unit	6000
4	2	/dev/cj2	Main file system	36000
4	3	/dev/cj3	Spare section	36000
4	4	/dev/cj4	Spare section	36000
4	5	/dev/cj5	Virtual floppy	2000
4	6	/dev/cj6	(cj3,cj4)	72000
4	7	/dev/cj7	Entire disk	11956
		ck		
4	0	/dev/ck0	Backup root	1988
4	1	/dev/swap	Swapping unit	8000
4	2	/dev/ck2	Main file system	72000
4	3	/dev/ck3	Spare section	72000
4	4	/dev/ck4	Spare section	72000
4	5	/dev/ck5	Virtual floppy	8000
4	6	/dev/ck6	Floppy	2000
4	7	/dev/ck7	Entire disk	238670

M	m	Name	Use	# Of Blocks
cm				
4	0	/dev/cm0	Backup root	1988
4	1	/dev/swap	Swapping unit	12000
4	2	/dev/cm2	Main file system	72000
4	3	/dev/cm3	Spare section	72000
4	4	/dev/cm4	Spare section	36000
4	5	/dev/cm5	Virtual floppy	36000
4	6	/dev/cm6	Floppy	2000
4	7	/dev/cm7	Entire disk	232100
cq				
4	0	/dev/cq0	Backup root	1988
4	1	/dev/swap	Swap	2000
4	2	/dev/cq2	Root	20000
4	3	/dev/cq3	Spare	20000
4	4	/dev/cq4	Spare	20000
4	5	/dev/cq5	(cq2+cq3)	40000
4	6	/dev/cq6	Virtual floppy	4000
4	7	/dev/cq7	Entire disk	69632
in				
14	0	/dev/in0	Backup root	1988
14	1	/dev/swap	Swap	20000
14	2	/dev/in2	Root	72000
14	3	/dev/in3	User	72000
14	4	/dev/in4	Spare	72000
14	5	/dev/in5	(cq2+cq3)	72000
14	6	/dev/in6	Virtual floppy	72000
14	7	/dev/in7		72000
14	8	/dev/in8		72000
14	9	/dev/in9		72000
14	10	/dev/in10		72000
14	11	/dev/in11		72000
14	12	/dev/in12		2000
14	13	/dev/in13		2000
14	14	/dev/in14		2000
14	15	/dev/in15		2000

M	m	Name	Use	# Of Blocks
		xe		
13	0	/dev/xe0	Backup root	1988
13	1	/dev/swap	Swap	20000
13	2	/dev/xe2	Root	72000
13	3	/dev/xe3	User	72000
13	4	/dev/xe4	Spare	72000
13	5	/dev/xe5	(cq2+cq3)	72000
13	6	/dev/xe6	Virtual floppy	72000
13	7	/dev/xe7		72000
13	8	/dev/xe8		72000
13	9	/dev/xe9		72000
13	10	/dev/xel0		72000
13	11	/dev/xel1		72000
13	12	/dev/xel2		2000
13	13	/dev/xel3		2000
13	14	/dev/xel4		2000
13	15	/dev/xel5		2000
		8" Floppy Disks		
4	224	/dev/zcfs0	zero offset s/s	500
4	225	/dev/zcfdssd0	zero offset d/s	1001
4	226	/dev/cfs0	one offset s/s	494
4	227	/dev/cfdssd0	one offset d/s	994
4	228	/dev/cfssdd0	one offset s/d	988
4	229	/dev/cfd0	one offset d/d	1989
		5 1/4", 48 tpi Floppy Disks [5000 series]		
4	208	/dev/cms0	s/d	160
4	209	/dev/cmdssd0	d/s	320
4	210	/dev/cmssp0	s/s	180
4	211	/dev/cmdssdp0	d/s	360
4	212	/dev/cmssdh0	s/d	320
4	213	/dev/cmdh0	d/d	640
4	214	/dev/cmssddhp0	s/d	360
4	215	/dev/cmdhp0	d/d	720
4	216	/dev/cmssdd0	s/d	320
4	217	/dev/cmd0	d/d	640
4	218	/dev/cmssddp0	s/d	360
4	219	/dev/cmdp0	d/d	720

M	m	Name	Use	# Of Blocks
5 1/4", 96 tpi Floppy Disks				
4	200	/dev/chs0	s/s	520
4	201	/dev/chdssd0	d/s	1040
4	202	/dev/chssddh0	s/d	1040
4	203	/dev/chdh0	d/d	2080
4	204	/dev/chssdd0	s/d	1200
4	205	/dev/chd0	d/d	2400
4	206	/dev/chssddp0	s/d	1280
4	207	/dev/chdp0	d/d	2560

Minor device numbers used for operating the 96 tpi drives in 48 tpi compatibility mode

4	176	/dev/cms0	s/d	160
4	177	/dev/cmdssd0	d/s	320
4	178	/dev/cmssp0	s/s	180
4	179	/dev/cmdssdp0	d/s	360
4	180	/dev/cmssdh0	s/d	320
4	181	/dev/cmdh0	d/d	640
4	182	/dev/cmssddhp0	s/d	360
4	183	/dev/cmdhp0	d/d	720
4	184	/dev/cmssdd0	s/d	320
4	185	/dev/cmd0	d/d	640
4	186	/dev/cmssddp0	s/d	360
4	187	/dev/cmdp0	d/d	720

M	m	Name	Use
1/4" Streaming Tape [5000 series]			
4	224	/dev/cs0	first partition file
4	225	/dev/cs1	second partition file
4	226	/dev/cs2	third partition file
4	227	/dev/cs3	fourth partition file
4	228	/dev/cs4	fifth partition file
4	229	/dev/cs5	sixth partition file
4	230	/dev/cs6	seventh partition file
4	231	/dev/cs7	eighth partition file

Index

.EDMAC 64
.globals 64
/bin 83
/dev 83
/dev/null 53
/doc 83
/etc 83
/etc/conf 87, 108
/etc/group 61
/etc/passwd 60
/etc/printers 51
/etc/startup file 15, 54, 105
/etc/ttys file 45
/include 83
/mnt 83, 85
/tmp 83
/usr 84
makedev 32

-A-

admin 60, 61, 62, 64, 83, 105, 106, 108, 128
administrator's functions 118
administrator's user id 60
auto-recovery 94
auto-start 94

-B-

backing up disks on tape 114
backup root 128
bad block map 128
bad tracks 71
bit map 72, 84, 128
block 128

block bit map 15
boot device 131
bootstrap program 71
bootstrap routine 131
buffered mode 131

-C-

cache disk blocks 89
carriage return 132
cc command 104
chown command 64
cnf__balloc 96
cnf__clock__rate 93
cnf__cmax 80, 96
cnf__cmin 80, 96
cnf__contig 96
cnf__dsmem 95
cnf__erase__kill 93
cnf__errlog__size 94
cnf__fill1 90
cnf__filler3 93
cnf__filler4 93
cnf__iawait__pri 92
cnf__iop__pollrate 94
cnf__ipcmsgsz 95
cnf__ipcnques 95
cnf__ipcqsize 96
cnf__ipctotmsg 95
cnf__logsize 92
cnf__maxlu 90
cnf__ndsegs 91
cnf__neventcalls 91
cnf__nlopen 91
cnf__nfsys 92
cnf__nlocks 93
cnf__nmulti 93
cnf__nptys 94
cnf__nsysbuf 89
cnf__numproc 90
cnf__pause__at__boot 93
cnf__quantum__pri__factor 92

cnf__readonly__root 94
cnf__recovery__rootdev 94
cnf__rootdev 89
cnf__script 94
cnf__start__ceil__pri 90
cnf__swapdev 89
cnf__swapsize 89
cnf__swap__priority 90
cnf__system__id 93
cnf__ttybufsize 92
cnf__use__auto 94
command arguments 131
Command Interpreter 62, 105
confedit command 98
configuration file 131
conftable command 102
control characters 132
cpw command 61

-D-

debe command 109
delete command 64
dev directory 133
device controllers 133
device Major/minor number 5
device nodes 133
device switch table 5, 133
devices 1
devices (buffered or unbuffered) 82

-E-

email command 61
error messages 132
eventcounts 83, 91

-F-

FD 72
fd bit map 84
file (gargantuan) 75

file (huge) 75
file (large) 74
file (small) 73
file contiguous 76
file descriptors 15, 72, 81, 84
file locking 93
file sizes 73
file system 133
file system header 65
file system logical 89
file types 80
find command 64
floppy 134
floppy formats 35
floppy image 134
free blocks 76
fscheck command 90

-G-

group 61

-H-

home directory 62

-I-

input file 134
installation command 109
interleave factor 134
IOP 131
ipcsend() 95

-K-

Korn Shell 128

-L-

locking (process) 104
logical devices 134

logical units 90, 107
login name 61
lpd command 53
lpmaint command 54

-M-

magic word 134
mailbox 62
major device number 134
major device numbers 5
Major Minor numbers 133
makefs command 15, 84
match command 104
minor device number 134
modem 55
modem and UniverseNet 57
modem dialing into onsite modem 56
modem hints 58
modem offsite set up 56
modem onsite set up 55
modem signing off 58
modem types 55
modem using 55
modifier 131
mount 135
mounted file systems 65, 85, 92
mount command 85

-N-

network 61
nice command 106
notify command 107

-O-

operating system 135
output file 135

-P-

package command 7, 15, 17, 109
parameters configuration 87
partition record 71, 135
partitioning 19
partitions 7
password 61
password file 60
pathname 135
peripheral 135
physical device 135
physical devices 134
pipe 82
print_partition command 20
priority 90, 92, 104
privileged user 128
ps command 107
PVH 7, 135

-Q-

quarter-inch tape 114

-R-

root 65, 135
root device 136
root directory 82
root recovery 94

-S-

setpri command 106
setuser command 117
setuser command 136
shared text 91
shell 62
sort command 106
space command 90
subdirectories [file system 65

swap device 136
swapping 104
sync() subroutine 89
sync command 72
sys 83
system configuration file 87
system console 132
system prompt 131

-T-

tape (half-inch) 36
tape (labelled) 37
tape (quarter-inch) 36
tape (unlabelled) 37
ted command 64, 104, 105
tp command 38
ttymodes command 52, 54, 93

-U-

unboot 137
unbuffered mode 137
unbug 136
UniverseNet 61
unmount command 86
UNOS Commands Manual 128
user id 61
user partition 15
User's Guide to UNOS 128
using tapes for back up 114

-V-

virtual floppies 7
virtual floppy 137
VME IOP Major Device Numbers 28

-W-

who command 104
write command 61

write_partition command 18, 20

