UniFLEX®
System Manager's Guide

technical systems consultants, inc.

/etc/tune

Change the specified parameters in a file containing a copy of the UniFLEX Operating System.

SYNTAX

```
/etc/tune <file_name> [<param_list>]
/etc/tune <file_name> [+r]
```

DESCRIPTION

The "tune" command is used to alter certain parameters which govern the behavior and performance of the UniFLEX Operating System. For a complete discussion of the use of this command, see the System Manager's Guide.

The command operates in three modes: read-only, interactive, and automatic. In read-only mode "tune" displays the current value, in the specified file, of each the of parameters it can alter. If the user does not have write permission in the specified file, "tune" automatically executes in read-only mode. A user who does have write permission in the file can execute "tune" in read-only mode by specifying the 'r' option.

If the user has write permission in the specified file and specifies neither a parameter list nor the 'r' option, "tune" executes in interactive mode. In this mode it displays current values one by one. To change the value of a parameter the user enters the new value and a carriage return following the display. To leave the value as is the user types just a carriage return. The "tune" command imposes certain restrictions on the values of the parameters it alters. It also imposes a limit on the amount of memory collectively used by the following parameters: buffers, iolists, files, locked_recs, mounts, and tasks. If the user enters a value in interactive mode which violates one of these restrictions, "tune" does not alter that value. Rather, it responds with an error message and waits for the user to enter a new value.

Instead of going through the entire list of parameters interactively, the user can directly specify on the command line which values "tune" is to change. The value of a parameter not mentioned on the command line does not change. After it makes the changes, "tune" displays a list of all parameters and the values specified by the user. If any of the values violates the restrictions mentioned previously, "tune" displays a

(continued)

message to that effect and sends a bell (control-G) to the terminal. Although the display shows the values specified by the user, "tune" does not make changes in the file if the changes violate the restrictions.

Arguments

<file_name>	The name of a file that contains a copy of the operating system.
<param_list>	A list of the parameters to change and the values to assign to them.

Format for Arguments

<param_list> <param_name>=<decimal_number>

Table 1 lists the parameters that "tune" can change.

Table 1. Parameter names and descriptions

<param_name>	Description
buffers	Number of system buffers.
iolists	Maximum number of lists of I/O characters.
DST	Flag for the observation of Daylight Savings Time (0 indicates it is not observed locally; 1, that it is).
files	Maximum number of files that can be open at one time.
locked_recs	Maximum number of entries allowed in the table of locked files.
mounts	Maximum number of devices that can be mounted.
pipe_dev	Device number of the pipe device.
root_dev	Device number of the root device.
seek_rate	Seek rate of the floppy disk drive.
swap_dev	Device number of the swap device.
tasks	Maximum number of tasks the system can simultaneously execute.
text_segs	Maximum number of unique shared-text programs that the operating system can simultaneously execute.
time_zone	Time difference in minutes between local time and Universal Time. A positive value of "time_zone" indicates the number of minutes west of Greenwich; a negative value, the number of minutes east.
user_tasks	Maximum number of tasks allowed to each user.

The values for all these parameters are originally set when one of the "/uniflex" files is copied from the master disk to a system disk. These default values, as well as the limits imposed on each parameter, are

(continued)

shown in Table 2.

Table 2. Default Values and Limits for "tune" Parameters

<param_name>	Default	Minimum	Maximum
buffers	sd	8	64
iolists	sd	16	255
DST	0	0	1
files	sd	16	255
locked_recs	32	0	Value of "files"
mounts	5	2	32
pipe_dev	sd	0	Last block device number
root_dev	sd	0	Last block device number
seek_rate	0	0	sd
swap_dev	sd	0	Last block device number
tasks	sd	8	64
text_segs	20	2	20
time_zone	300	-1440	1400
user_tasks	10	5	Number of tasks

Notes: sd = system-dependent

Options Available

r Execute the command in read-only mode--that is, display the current value for each parameter, but do not make any changes.

EXAMPLES

1. /etc/tune /uniflex +r
2. /etc/tune /usr2/uniflex2 tasks=32 swap_dev=1

The first example displays a list of the items that "tune" can adjust. The current value of each item in the file "uniflex" in the root directory appears in parentheses.

The second example changes the specified parameters in the file "uniflex2" (the mini-Winchester version of the operating system) in the directory "/usr2". Presumably, a system disk is mounted on "/usr2". This command sets the maximum number of tasks allowed on the system to 32 and defines floppy drive 1 as the swap device. In order for this particular version of the operating system to be able to perform swapping, a floppy disk formatted with swap space must be in floppy

(continued)

drive 1. The disk should not be mounted.

NOTES

- The "tune" command changes the values of the parameters in the file specified on the disk only, not in memory. Therefore, the changes have no effect until the user boots the operating system from the modified version.

ERROR MESSAGES

'r' option incompatible with command-line parameters.
The 'r' option, which tells "tune" to operate in read-only mode, conflicts with the specification of parameters on the command line. The command is aborted.

***Value out of range [num_1, num_2]
The value specified for a parameter is not within the range of acceptable values. The limits of the range are shown inside the square brackets.

***Value must be a multiple of <num>.
The value for the number of buffers in the system must be a multiple of 8. The value for the time zone must be a multiple of 60.

***Total system table space exceeded.
The specified change violates the restriction on the amount of memory collectively allowed to buffers, iolists, files, locked_recs, mounts, and tasks.

SEE ALSO

System Manager's Guide

UniFLEX®
System Manager's Guide

MANUAL REVISION HISTORY

Revision	Date	Change
A	6/85	Original Release, System Manager's Guide for 6809 UniFLEX

DISCLAIMER

The supplied software is intended for use only as described in this manual. Use of undocumented features or parameters may cause unpredictable results for which Technical Systems Consultants, Inc. cannot assume responsibility. Although every effort has been made to make the supplied software and its documentation as accurate and functional as possible, Technical Systems Consultants, Inc. will not assume responsibility for any damages incurred or generated by such material. Technical Systems Consultants, Inc. reserves the right to make changes in such material at any time without notice.

Contents

Contents

Contents

Contents

Preface

This manual is intended to aid the system manager for a UniFLEX® 6809 Operating System in the care and operation of the system. The document pertains only to the operating system itself, not to any of the support software sold separately. Thus, for example, an unqualified reference to a spooler in this manual refers to the standard spooler which is sold with the operating system, not to the more sophisticated Enhanced Printer Spooler. You should thoroughly read the documentation for any support software own before trying to put it on your system.

Preface

Chapter 1

Starting the System

1.1 Introduction

This chapter deals with the day-to-day operation of the UniFLEX Operating System. Much of the material covered here is information you need every time you bring up your system.

1.2 Booting the System

Before you can use the UniFLEX Operating System you must execute the boot program, which resides in the monitor ROM (read-only memory). The boot program loads a particular sector, the boot sector, from the device specified by the user into main memory. It then begins execution of the code contained in that sector. This code loads the executable file "/uniflex" into main memory and starts to execute it. The file "/uniflex" performs several initializations, then starts the first task which, in turn, executes a single-user shell program. The shell program sends a system prompt to the screen, indicating that the system has been successfully booted and that the operating system is ready to work for you. This process, which is known as booting the system, varies from system to system. It is discussed in detail in Getting Started with UniFLEX.

When you have successfully booted the system, the following message appears on your screen:

```
UniFLEX Operating System
Copyright (C) 1983 by
Technical Systems Consultants, Inc.

Version <version_num>   Released <release_date>
Configuration:  <configuration_information>

Total user memory = <memory_left_after_loading_UniFLEX>

++
```

Record the version number and the release date in a safe place. Also

record the serial number of the computer and the serial number of your copy of the UniFLEX Operating System (located on the label on the master disk). If you ever need to contact either the manufacturer of your hardware or Technical Systems Consultants, you will need this information.

The amount of memory left after loading the operating system depends on the particular configuration of the computer and the amount of memory installed in it. A small operating system may consume as little as 48K; a larger system, 64K or more.

The system prompt at the end of the message indicates that the operating system is ready to accept commands. At this point the system is in single-user mode. In single-user mode only one terminal--tty00, also known as the console--is active.

Single-user mode is used at times when you want to be certain that you are the only user on the system. Such times include the time immediately after booting the system when you must perform certain daily operations, and times when you are performing system maintenance.

1.3 Setting the Date

The first thing you should do after booting the system is to set the date and time. Some hardware comes equipped with a built-in clock from which the operating system can read the date and time. Other systems require you to set the date and time yourself.

1.3.1 Systems with a Built-in Clock

Three kinds of hardware are currently equipped with a built-in clock: all hardware from Pennywise Peripherals; GIMIX III systems; and systems from Southwest Technical Products that feature a peripheral processor interface (PPI). To set the date and time on these machines you must have the operating system in single-user mode. The format for the command on Pennywise equipment is

```
rtc [+s]
```

The format for the command for the other two systems is

```
set_tod [+s]
```

It may help you to remember the names of the commands if you know that "rtc" stands for "real time clock"; "set_tod", for "set time of day".

In either case the use of the command without the ´s´ option sets the date and time from the built-in clock. If for any reason you want to set the date and time yourself, you should use the ´s´ option. In response to this option the operating system prompts you for the following: the year (a two-digit number assumed to be in the twentieth century); the month (a number from 1 to 12 inclusive); the day (a number from 1 to 7 inclusive where Sunday is represented by the number 1, Monday by the number 2, and so forth); the hour (a number from 0 to 23 inclusive); the minute (a number from 0 to 59 inclusive); and seconds (a number from 0 to 59 inclusive).

1.3.2 Systems without a Built-in Clock

The "date" command for machines without a built-in clock has two forms: one with an argument and one without. Any user may execute the "date" command without an argument. In response, the system returns the current date and time. The system manager may also use the "date" command with an argument. The syntax for the command in this case is as follows:

```
date [<mm>-<dd>-<yy>] <hr>:<min>[:<sec>]
```

where <mm> is a number from 1 to 12 inclusive representing the month, <dd> is a number from 1 to 31 inclusive representing the day, and <yy> is a two-digit number representing the last two digits of the year. The time must be 24-hour-clock time where <hr> is a number from 0 to 23 inclusive representing the hour, <min> is a number from 0 to 59 inclusive representing minutes, and <sec> is a number from 0 to 59 inclusive representing seconds. For example, the following command sets the date to 7:30 AM on April 17, 1985:

```
date 4-17-85 7:30:00
```

If the system has only been down a short time, it may not be necessary to set the month, day, and year. If you do, you may include just the day, the day and the month, or the day, month, and year. The operating

system takes values for the day, month, and year from the disk if you do not specify them.

It is, however, always necessary to set the time (if you omit the seconds argument, the system assumes a value of 0). If you do not specify a time, the system responds with the message

```
Command syntax error
```

Even though it is not always necessary, it is a good idea to specify both arguments to the "date" command. Because other parts of the operating system reference the date, it is important for the date to be correct. The consequences of having the date set incorrectly range from a minor inconvenience to a serious problem.

You should enter the local time when you set the date. The system stores the time internally in seconds since January 1, 1980, at the zeroth meridian (in Greenwich, England). As it makes this conversion, it adjusts the result for the local time zone and for Daylight Savings Time if it is in effect (see Section 6.3.3).

1.4 Message of the Day

Whenever a user logs in, in response to the login prompt, the login program reads the file "/etc/log/motd" and sends it to standard output before issuing the system prompt. Thus, if you want to send a message to all users, you can enter it in this file.

You can edit the message-of-the-day file just as you would edit any other file. An example follows:

```
++ chd /etc/log
++ edit motd +b
^d!
#i
    1.00=April 18, 1985:  A new Pascal compiler was
    2.00=installed on the system today.  Documents which
    3.00=describe its use are available in the main office.
    4.00=All users are welcome to use this new compiler.
    5.00=#s
++
```

The first line of this example changes your working directory from the root directory (your default location on booting) to the directory "/etc/log". The next line invokes the editor, telling it to edit the file "motd" without creating a backup file. The next command deletes the contents of the entire file. (You may not always want to delete old messages. In such a case simply leave out the ´^d´ and position yourself at the bottom of the file with the command ´!´ instead.) The letter ´i´ in response to the editor´s prompt puts you in insert mode. Now you can enter your message.

It is a good idea to date your messages so that both you and the users know when they were first issued. When you finish typing your message, you can exit the editor by typing a pound sign, ´#´, as the first character on a line, followed by an ´s´ for "stop". (See 6809 UniFLEX: A Tutorial and the documentation on the editor in The UniFLEX Operating System for more detailed instructions on the use of the editor.)

1.5 Formatting a Disk

Any disk that is to be used with the UniFLEX Operating System must first be properly formatted. The operating system supports commands which format floppy disks and mini-Winchester hard disks. Please note that Technical Systems Consultants does not supply software to format a CDS Winchester disk. If you have such a disk, you must obtain the necessary formatting programs from Southwest Technical Products, Corp.

The "format" commands not only establish the physical boundaries of the sectors on the disk but also set up the UniFLEX file system by writing both the boot sector and the SIR (system information record), by establishing the root directory, by initializing the file descriptor nodes (fdns), and by reserving swap space. The precise function of a "format" command can be altered by the many available options. These commands are documented with other UniFLEX utilities in the operating system manual.

If you are formatting a disk that is to be used as a system disk, be sure to use the ´r´ option to reserve some swap space. Swap space is a section of the disk that is reserved for storing tasks that are swapped out of memory to make room for tasks of higher priority. A system disk cannot function without swap space. The amount of swap space you need depends on the individual nature of your system--in what ways it is used and how heavily it is used. The following factors should be considered when determining the amount of swap space to reserve:

1. The largest amount of swap space a single 6809 task can use is 64K.

2. A system is usually running a number of tasks which may not be obvious to any users. Such tasks include the login program, the initialization program, and extra shell programs.

3. A compiler may call the assembler and loader. If it does so, it may cause the compiler to be swapped to the disk.

4. The swap space used for a particular task must be contiguous. Therefore, if you know that the part of the disk that will be used for swap space contains any bad spots, you should assign extra swap space to make up for the discontinuity.

5. The more random access memory (RAM) you have, the less swap space you should need.

6. The more users on your system, the more swap space you need.

7. The argument to the 'r' option specifies how may cylinders to reserve for swap space. A cylinder consists of all the data tracks that can be accessed by all the read-write heads without mechanically moving the head assembly. If you are using a Winchester disk, you can easily calculate the number of bytes reserved by each cylinder. The operating system always formats such a disk with 17 sectors per track and 512 bytes per sector. Thus, the number of bytes per track is

   17 sectors/track * 512 bytes/sector = 8704

   To convert to the number of bytes per cylinder, you must multiply by the number of tracks per cylinder. For a hard disk the number of tracks per cylinder is equal to the number of heads in the disk.

   It is impossible to give a single amount of swap space that serves all systems. However, Table 1-1 shows amounts which seem to work well for most systems. If your system disk is a floppy disk, it is best to use a double-sided, double-density disk. Double-sided, single-density disks and

single-sided, double-density disks may be used as system disks; however, you may have to reserve close to half the storage space on the disk for swap space. Your system disk should not be a single-sided, single-density disk. The operating system alone nearly fills such a disk.

Table 1-1. Suggested Guidelines for Amount of Swap Space

Type of Disk	Sides	Density	Cylinders to Reserve
Floppy disk	1	Double	30
Floppy disk	2	Single	30
Floppy disk	2	Double	15
Hard disk	---	---	15-20

1.6 Sequestering Bad Blocks

It is almost inevitable that some parts of a disk become physically damaged. If the operating system tries to read from or write to such a "bad block", it fails and sends the user an I/O error. Depending on the location of the bad block, an I/O error may range from a minor inconvenience to a serious problem that causes the system to crash. It is therefore wise to remove a bad block from use as soon as it is detected. The UniFLEX Operating System supports two methods of doing so: one during the formatting of a disk; the other after the disk has been formatted. In either case the bad block is placed in a file named ".badblocks" in the root directory (also known as the bad-blocks file). The operating system knows not to allocate any of the blocks in this file.

1.6.1 Formatting with the 'l' or 'L' Option

The "format" commands support two options which allow you to specify that certain blocks should be placed in the bad-blocks file. The use of these options is described in detail with the documentation for "format" in the operating system manual. Basically, you should look at the list of bad blocks supplied by the manufacturer of the disk. This list

should specify a head, a cylinder, and either a range or a sector number for each bad block. The "format" command must be told the head, cylinder, and logical 512-byte sector of each bad block. The tables in Appendix B and Appendix C of the format documentation enable you to convert the information supplied by the manufacturer to the information needed by the operating system.

If you know of additional bad blocks on the disk, you should specify them to the "format" command as well. The "devcheck" command, which searches a disk for bad blocks, reports the address of each bad block as a decimal number. You must convert these addresses to the logical, 512-byte sector required by the format command. To make the conversion you must know the address reported by "devcheck", the number of sectors per track, and the number of tracks per cylinders on your disk. For a hard disk the number of sectors per track is always 17; the number of tracks per cylinder is equal to the number of heads in the disk. For a single-density, eight-inch floppy disk the number of sectors per track is 8; for a double-density one, 16. For a single-density, five-inch floppy disk the number of sectors per track is 5; for a double-density one, 10. For any floppy disk the number of tracks per cylinder is equal to the number of sides available for storage: 1 for a single-sided disk; 2 for a double-sided disk.

To make the conversion perform the following procedure:

1. Divide the address supplied by "devcheck" by the number of sectors per track. Add 1 to the remainder to determine the logical sector number.
2. Take the quotient from the previous division operation and divide it by the number of tracks per cylinder. The new quotient is the cylinder number; the new remainder is the head number.

For example, if the address of a bad block on a mini-Winchester with six heads is 10541, the conversion is as follows:

1. 10541/17 = 620, remainder = 1
   Adding 1 to the remainder gives the logical sector number, 2.

2. 620/6 = 103, remainder 2
   Thus, the cylinder number is 103; the head number, 2.

You now have all the information you need for specifying that block to the '1' or 'L' option.

1.6.2 The "badblocks" Command

Once a disk has been formatted, it is not necessary to reformat it every time you need to add blocks to the bad-blocks file. Instead, you can use the "badblocks" command. This command, which is described in the operating system manual, reads a list of decimal addresses from the command line and places the corresponding blocks in the bad-blocks file.

1.7 Verification of the Medium

Whenever you boot your system, it is wise to make certain that the structure of the disk or disks you are using is intact. Any errors are best detected before you start to use the system. The logical structure of the disk can be checked with the utility "diskrepair" (see Chapter 7). If you suspect a physical problem, you should execute the "devcheck" command (see The UniFLEX Operating System), which checks the physical integrity of every block on the disk.

1.8 Mounting Devices

When you boot your system, you are in the root directory on the root device. The root directory is the directory at the top of the directory tree. It has no parent directory; all other directories are descendants of the root directory. The root device is usually a hard disk.

You may need to use devices other than the root device. For instance, there may be information on another disk which users need to access. You can make such a device accessible by mounting it on a node of the directory tree. Only the system manager may execute the "mount" command. A command of the form

/etc/mount <dev_name> <dir_name>

mounts <dev_name> at the directory <dir_name>. As long as the device is mounted, any references to <dir_name> actually access the root directory of the device mounted there. Any files in the directory at which the device is mounted are inaccessible as long as the device is mounted.

A block device can be mounted on any directory in the file system. However, because any files in a directory on which a device is mounted are inaccessible, it is good practice to mount a device on an empty directory. For convenience, it is also a good idea to mount the device on a directory in the root directory, in order to minimize the length of the reference to the mounted device. The operating system comes with an empty directory called "usr2", which resides in the root directory. This directory is a convenient node on which to mount devices. You can create additional directories for this purpose if you need them.

For instance, suppose you have a file called "billing_info" in the root directory of a removable-cartridge disk, "rc0", and that users regularly need access to the information in this file. You can mount the disk by placing it in drive "rc0" and invoking the following command:

```
/etc/mount /dev/rc0 /usr2
```

This command tells the operating system to place the device "/dev/rc0" at the node "/usr2" in the root device. Users can now reference this file as "/usr2/billing_info".

The "mount" command supports one option, 'r', which tells the system to mount the device for reading only. Use of this option allows users to read files on the mounted device but prohibits them from writing to the device. Note that contrary to the usual UniFLEX format, you do not use a plus sign before the letter 'r' when invoking this option.

A mounted device must remain in the disk drive until it is unmounted with a command of the following form:

```
/etc/unmount <dev_name>
```

If a mounted disk is removed by accident, it should be reinserted, unmounted, and checked for damage with "diskrepair" (see Chapter 7).

1.9 The History File

The root directory contains a directory called "act", which is initially empty. This directory is a place to store accounting information. (The additional software package, User/System Accounting System, makes extensive use of this directory.) As shipped, the only accounting

procedure the UniFLEX Operating System performs is the maintenance of a history file, "/act/history". If this file exists, the operating system automatically makes an entry in it each time the system is booted or stopped, each time it goes from single-user mode to multi-user mode or from multi-user mode to single-user mode, each time the system manager sets the date, and each time any user logs in or out.

A listing of the history file is unintelligible. If you want to extract information from the file, you must execute the "history" command. The output from this command is explained in the documentation of the UniFLEX Utilities in the operating system manual.

If you want to maintain a history file, all you need to do is create a file named "/act/history" with the following command:

```
create /act/history
```

After creating the file you should alter the permissions so that other users cannot accidentally destroy it by creating a file with the same name. To do so, you must deny other users write permission in the file:

```
perms o-w /act/history
```

The history file can rapidly grow and occupy a lot of space on your system disk. Therefore, you may want to start a new history file periodically. If you want to save the old information, you can copy the file to a backup device. It is a good idea to rename the file in some way so that the name indicates the period of time covered by that particular file. For instance, if you start a new history file once a month, you may append the name of the month to the file as you back it up. When you have backed up the file, simply create the file "/act/history" again. This command truncates the existing file of that name to length 0.

1.10 Using a Printer Spooler

The UniFLEX Operating System is shipped with a general-purpose printer-spooler which must be configured before use. A printer spooler coordinates the printing of files in response to requests by users.

1.10.1 Configuring a Printer Spooler

Before you can use the general spooler you must do three things: link it to an appropriately named command; create a directory to contain the files you send to the spooler; and initiate the spooler.

1.10.1.1 Creating a spooler command

In order to create a spooler command, you must select the appropriate device driver. At least two printer drivers are provided with the standard version of the UniFLEX Operating System: "spr", for a serial printer, and "ppr", for a parallel printer. Both drivers reside in the directory "/dev". Consult the documentation on hardware set-up for the details of your particular standard drivers.

Suppose you want to use a serial printer. Obviously, you would select "/dev/spr" as the device driver. The command you create to send things to the spooler is simply a link to the executable file "/etc/print", which is a part of the UniFLEX Operating System. However, the last component of the name of file to which you link "/etc/print" much match the last component of the name of the device driver ("spr"). Since many people may be using the command, it is convenient to place it in the directory "/bin" so that the users do not have to specify the full file specification or the path name every time they invoke the spooler. You can accomplish all this with the following command:

```
link /etc/print /bin/spr
```

which puts into the directory "/bin" an entry which points to the file "/etc/print".

1.10.1.2 Creating a spooler directory

The next step is to create a directory for the spooler. The last component of the name of this directory must also match the last component of the name of the device driver, and the directory must reside in the directory "/usr/spooler". This is simply done:

```
crdir /usr/spooler/spr
```

In general it is wise to set the permissions in this directory so as to deny access to all users except the system manager. The following command does so:

```
perms o-rwx /usr/spooler/spr
```

Your system now contains a new command, "spr", which spools files to the serial printer. If you want to change the name of the command to something more memorable, such as the an abbreviation of the name of the manufacturer of the printer, you can do so by renaming or linking the device driver to "/dev/<new_name>", renaming or linking the executable file "/bin/spr" to "/bin/<new_name>", and creating a directory of the appropriate name in "/usr/spooler" (you should not rename or link a directory). If you choose to link rather than to rename the device driver and the executable file, you must be careful to avoid having different users accessing the same device by different names at the same time. If this should happen, the files may be sent to the same device simultaneously, resulting in the intermingling of the files as they are printed.

1.10.1.3 Initiating a printer spooler

Now you have a spooler command and a spooler directory, but before you can actually execute the command, you must initiate the spooler. In fact, each time you boot the system, you must initiate each standard spooler you want to use. (You can simplify this task by putting the relevant commands in the file "/etc/startup"--see Section 1.14.) A spooler is initiated by a command of the following form:

```
/etc/insp <splr_name> [+f]
```

where <splr_name> must match the last component of the file specification of the corresponding device. The 'f' option suppresses the banner page and the form-feed character which are both normally printed before each print job. In this example the appropriate command is

```
/etc/insp spr
```

When you execute this command, the system creates a background task, which runs continuously until you deliberately stop it with a "pstop" command. In addition, it creates a file in the spooler directory called ".mrk*splr?", which contains the task ID of the background task. The background task checks the contents of the spooler directory approximately every 20 seconds. It sends any files present to the

printer and deletes them from the spooler directory. Note that the original copies of the files are still intact; only the references to them in the spooler directory are deleted.

1.10.2 Using the Spooler Command

To use this spooler a user simply types

```
spr <file_name>
```

When a user invokes the spooler command, the command generates a name for the file being spooled. It does so by appending a three-digit number to the user name of the person who invoked the spooler. Once it has created the new name, the spooler command makes an entry in the corresponding spooler directory. This entry is like any other entry in a directory; it contains a pointer to the file descriptor node (fdn) of the file being spooled and the newly generated name of the file. If the file is on the same disk as the spooler directory and it is sent directly to the spooler, the entry in the spooler directory is simply a link to the original file. If the file is on another disk, the operating system makes a copy of the original file on the same disk as the spooler directory and makes an entry for that copy of that file in the spooler directory. Finally, if the user sends the file through a pipe, the spooler command copies the output from the pipe into a file and makes the corresponding entry in the spooler directory.

One consequence of whether or not the entry in the spooler directory is a link to an existing file is that the linking process causes the date and time of the last modification of the file to change whereas the copying process does not. Thus, the spooler command updates the date and time of a file that is on the same disk as the spooler directory and is sent directly to the spooler. A user who wishes to avoid changing the date and time can always send the file to the spooler through a pipe using the following command:

```
list <file_name> ^ <splr_name>
```

1.10.3 Shutting Down a Printer Spooler

Under normal conditions you should initiate a standard printer spooler each time you boot the system. Before shutting down the system you should send a "pstop" command to each active spooler. The spooler finishes printing any active job before it stops. The syntax of the "pstop" command is

pstop <splr_name>

The "pstop" command, like the "/etc/insp" command, acts on only one device at a time. You must issue a separate command for each spooler. In response to a "pstop" command the system sends an interrupt to the background task created by the "/etc/insp" command. The background task finishes printing the current task (if one exists), then deletes the ".mrk*splr?" file in the appropriate spooler directory.

If you forget to issue the "pstop" command before shutting down the system, or if the system crashes, the spoolers may be in a peculiar state. When the system goes down, through normal shut-down procedure or through a crash, all tasks are, of course, interrupted. Thus, even if you do not issue a "pstop" command, the background tasks associated with each spooler disappear whenever the system shuts down. However, the ".mrk*splr?" files are not automatically deleted when the system shuts down. You may, therefore, find yourself in the situation of having ".mrk*splr?" files, but no background tasks associated with the task IDs the files contain. If you try to issue the "/etc/insp" command in such a case, the operating system responds

File ´.mrk*splr?´ exists - spooler not invoked.

If this happens, simply issue a "pstop" command to the spooler in question. In such a case, because there is no background task to receive the interrupt or to delete the appropriate ".mrk*splr" file, the "pstop" command assumes the responsibility of finding and deleting the troublesome file. Once this file is deleted, the "/etc/insp" command can successfully invoke the spooler.

1.10.4 Summary of Routine Spooler Use

The following table summarizes the steps in making a printer spooler functional.

Table 1-2. Spooler-related Tasks for the System Manager

Task	Frequency	Command
Create command	Once	link /etc/print /bin/<splr_name>
Create directory	Once	crdir /usr/spooler/<splr_name>
Activate spooler	Every boot	/etc/insp <splr_name>
Terminate spooler	Every shutdown	pstop <splr_name>

1.10.5 Repairing a Damaged Printer Spooler

Occasionally, the necessary links between the files that control your printer spoolers may be broken. This problem is most likely to manifest itself when you update the spooler program. You may find, after updating your system disk, that the bugs which were supposed to have been fixed are not fixed, or that the enhancements which were supposed to have been made are not on your system.

In either case you should first check to see if the links that are essential to the functioning of the spooler have been broken. You can establish this by doing a "dir +l" on the directory "/etc". The entries for the files "prcon" and "print" should look like this (of course, the date and time differ from those shown here):

```
prcon      2     6     rwx---  bin     14:14 Apr 16 1985
print      3     <N>   rwx---  bin     14:14 Apr 16 1985
```

The link count for each file is the second number in the entry. The link count for "prcon" should always be 6. The link count for "print", <N>, varies depending on the number of spooler commands you create. The number <N> should be 1 plus the number of spooler commands you have linked to that file.

If either link count is incorrect, you should completely remove the spooler from the system and recreate it from the master disk. The following series of commands deletes the necessary files:

```
chd /etc
kill insp prcon print
chd /usr/bin
kill end idle next pstop purge rerun
```

In addition you must delete any spooler commands that you created. These commands are the ones in "/bin" that you linked to "/etc/print". It may be wise also to delete the spooler directories such as "/usr/spooler/spr". Before you can delete them with the "kill +d" command, you need to delete all the files they contain.

Now that you have completely removed the old spooler from your system, mount your master UniFLEX disk as "/usr2". The following steps rebuild the spooler:

```
chd /usr2/etc
copy insp print prcon /etc
chd /etc
owner system insp print prcon
chd /usr2/usr/bin
copy purge /usr/bin
chd /usr/bin
link /etc/prcon end
link /etc/prcon idle
link /etc/prcon next
link /etc/prcon pstop
link /etc/prcon rerun
```

This series of commands puts the standard spooler onto your system disk as it was originally supplied. You must now perform the usual steps, described earlier in this section, to create your own individual spooler commands and spooler directories.

1.11 Adding New Programs

When you add a new executable program to your system, you must decide what directory to put it in. The decision depends, in part, on who is to have access to the program. System-wide programs can conveniently go in one of two places--the directory "/bin" or the directory "/usr/bin". The shell program automatically searches both these directories when it is looking for an executable file (see 6809 UniFLEX: A Tutorial). Commonly used programs should go in "/bin" because the shell program searches that directory before it searches "/usr/bin". Less commonly used programs belong in "/usr/bin". It is wise to keep less than sixty

files in each of these directories. Having more files may result in a slight decrease in system response.

1.12 Maintaining Backup Files

One of the hazards of working with computers is that they do not always behave perfectly. Power failures and overheating are just two of the things which may cause a system to malfunction. Whenever a system shuts down in any but the prescribed fashion (see Section 1.15), the structure of the system disk and any disks mounted on the system may be damaged. Although it is often possible to recover completely from such an event (see Chapters 7 and 8), recovery is sometimes impossible. Some or all of the files on the system may be irreparably damaged.

Your best protection against disaster is the maintenance of a proper backup system, which consists of a copy of every file in use on your system. You will never know how important the maintenance of a good backup system is until you need, but do not have, one. It is a good idea to maintain two backup copies of everything and to store one copy at a different site. This precaution ensures you of an intact copy of the files even in the event of fire, flood, or some other catastrophe at your primary site.

You can make backup copies by using the "copy" command, which copies regular files only. Backing up a large number of directories with just the "copy" command is, however, extremely tedious and time-consuming. Two other programs are available which greatly simplify the task: "copy-dir" (in Utilities Package I) and "backup" (in Utilities Package II).

1.13 Going to Multi-user Mode

When you have completed your routine tasks and are ready to put the system into multi-user mode so that other users may log in, simply execute the following command:

```
log
```

1.14 The File "/etc/startup"

As you become familiar with the operating system, you will find that you want to perform certain tasks every time you boot the system. For instance, each time you boot the system you must initiate any printer spoolers that you wish to make available to the users. To avoid having to type such commands every time you boot the system, you can create a special file called "/etc/startup" and place the commands in this file. Each line of this file should consist of one UniFLEX command. You must then set execute permissions on the file:

```
perms u+x /etc/startup
```

When you type the "log" command, which takes the system from single-user mode to multi-user mode, the operating system looks for this file and, if it exists, executes all commands in the file before entering multi-user mode. If, however, any of the commands in the file fails, the operating system terminates execution of the file and puts the system in multi-user mode.

A sample start-up file, which initiates two printer spoolers, might look like this:

```
/etc/insp spr
/etc/insp ppr +f
```

1.15 Shutting Down the System

Shutting down the system from multi-user mode is a two-step process. You should always follow this procedure. Do not reset the computer unless forced to by a system crash. Failure to follow the shut-down procedure may result in damage to the system disk and any disks mounted on the system.

1.15.1 Step One

The "shutup" command, which only the system manager may execute, takes the system from multi-user to single-user mode. The format for this command is

/etc/shutup [[-]<minutes>]

By default, the "shutup" command takes the system from multi-user to single-user mode by sending a hang-up interrupt to all tasks. This interrupt is followed by a 15-second delay before the system actually enters single-user mode. If any tasks are being executed when the "shutup" command is run, the hang-up interrupt permits many of them (those that recognize and honor the interrupt) to terminate cleanly without the loss of any data. You may suppress both the interrupt and the 15-second delay by using the optional minus sign.

The argument <minutes> is a number between 0 and 60 inclusive, which specifies the number of minutes the system should wait before beginning to shut down. The default argument is 15.

When you execute the "shutup" command, its task number is sent to standard output so that you can subsequently terminate the "shutup" program with the "int" command if necessary. The "shutup" command itself sends a message announcing the impending shutdown to all terminals that are logged in (even if they have normal messages locked out). It repeats this message at intervals until the system actually shuts down. In the default case of fifteen minutes, the message is sent fifteen, ten, five, three, two, and one minute prior to shutdown.

1.15.2 Step Two

It is a good idea, once the system is in single-user mode, to unmount any mounted devices. Then, to bring the system to a halt, issue the following command:

/etc/stop

Only the system manager can issue this command. The system must be in single-user mode for the command to succeed.

At this point it is advisable to reset the system in order to withdraw the read-write heads from above the surface of the disk. When the Unibug prompt appears on your screen, shut off the power to the computer. Also shut off the power to any peripherals attached to the computer.

Chapter 2

The Password File

2.1 Introduction

As system manager you are responsible for maintaining the list of users, which is stored in the file "/etc/log/password". This file must contain an entry for each user.

2.2 Structure of the Password File

Each entry in the password file has following form:

<user_name>:[<password>]:<ID>:<home_dir>:[<login_program>]

To add a user to the system you must edit the password file and place an entry for that user in the file. This entry must contain a user name, a user ID, and a home directory. The other fields in the entry are optional.

2.2.1 User Name

A user name is the name that a user types in response to a login prompt in order to gain access to the system. It is best to assign a user name which actually identifies the person to whom it refers. Obvious choices are the user's first name, last name, or initials. The operating system imposes certain restrictions on a user name:

1. The name must consist entirely of lowercase letters.
2. The name must contain no more than eight characters.
3. The name must be unique.

The operating system cannot check the validity of a user name. However, it assumes that these restrictions have been honored. Violation of the

rules may, therefore, cause unpredictable results.

You assign a user name simply by editing the password file.

2.2.2 Password

A password is a safeguard against unauthorized access to the system. While a password is not essential, it is recommended that you assign one to each user. The user then has the option of changing the password by using the "password" command.

The operating system imposes no restrictions on passwords. The following guidelines, however, help ensure the security of the system:

1. Passwords should be five or six characters long. (Longer passwords are accepted, but in any case only the first sixteen characters are significant.)

2. Passwords should be a random mixture of letters and numbers.

3. The letters used should always be lowercase.

You assign a password to a user by using the "password" command with an argument. The syntax for the command is

password [<user_name>]

Only the system manager may use the optional argument. In response to this command the operating system asks for the password. When you type the password, the letters do not appear on the screen. They are deliberately suppressed in order to maintain the secrecy of the password. The operating system asks you to retype the password in order to verify the entry. If what you type the second time does not match what you typed the first time, it responds

```
Retry different - password unchanged.
++
```

If, however, you type the same password both times, the operating system enters an encrypted form of the password in the password file. Thus,

when a user lists the password file, it is obvious which users have passwords because their password fields are not empty. However, because the passwords are encrypted, it is not obvious what they are.

While it is operating, the "password" command creates and uses the file "/tmp/pswd". If this file already exists when you execute the "password" command, the operating system returns the following error:

```
System busy.
```

The file may be there for one of two reasons: someone else is using the "password" command or a previous use of the "password" command was interrupted (probably by a system crash) before it deleted the file. The safest approach is to ascertain whether or not someone else is using the file. If no one is using the file, issue the command

```
kill /tmp/pswd
```

It is important to be sure that no one is using the file when you execute this command. Otherwise, the password file itself may become corrupted. To minimize the likelihood of having this situation arise, you can delete all temporary files every time you take the system from single-user mode to multi-user mode by putting the following command in the file "/etc/startup" (see Section 1.14):

```
kill /tmp/*
```

If any user ever forgets his or her password, you must edit the password file and remove the corresponding encrypted password from the password field. The "password" command can then be used (either by the system manager or the user) to create a new password.

2.2.3 User ID

You must assign a user ID to each user. User IDs can be in the range from 1 to 32,000 inclusive. The system manager's ID must always be 0. The system comes with a second user, "bin", whose user ID is 1; however, that ID can be changed.

You assign an ID by editing the password file.

2.2.4 Home Directory

The fourth field in each entry in the password file contains the name of the user's home directory. The home directory is the directory that a user enters by default upon logging in. Each user must have a home directory. In general, the home directory is named "/usr/<user_name>". If no home directory is in the password file, the user receives the following message when attempting to log in:

No directory!

The operating system then sends another login prompt to the terminal.

You assign a home directory by editing the password file. However, you must also create the home directory using the following command:

crdir /usr/<user_name>

It is a good idea to make the user the owner of the home directory with the following command:

owner <user_name> /usr/<user_name>

Otherwise, the user cannot set the permissions for the home directory.

2.2.5 Login Program

The last field in an entry in the password file contains the name of the login program, which is the program that begins execution when the user logs in. If the login program does not exist, or if for any reason the operating system cannot execute the specified program, a message to that effect is sent to the terminal, followed by a login prompt. If the field is empty, the default is the shell program.

You assign a login program by editing the password file.

2.3 Original Password File

As shipped the UniFLEX Operating System has two entries in the password file:

```
system::0:/:
bin::1:/:
```

Each of these entries contains only the required fields; the optional fields are absent. Thus, as created, your system recognizes two users: "system" and "bin". Neither user has a password. The user ID for system is 0 (as it must be); the user ID for bin is 1. The home directory for both users is the root directory. The login program for both is, by default, the shell program.

One of your first tasks as system manager should be to assign passwords to these users. Use the "password" command to make these assignments.

2.4 Adding a User to the System

To add a user to the system you must make an entry in the password file that contains three things: a unique user name, a unique user ID, and a home directory. In addition you must be sure to actually create the appropriate home directory and to make the user its owner. It is also wise to assign a password to the new user.

Suppose you want to add a user with the user name "larry" to the system. First of all, you must make a new entry in the password file. Before adding the new user to the file, select a user ID which is not already in use. If you selected the number 100, you would add the following line to the password file (see 6809 UniFLEX: A Tutorial or the documentation on the editor in the operating system manual for information on editing a file):

```
larry::100:/usr/larry:
```

Next you should create the directory "/usr/larry" and make "larry" the

owner of that directory:

```
crdir /usr/larry
owner larry /usr/larry
```

Now the new user can log in. His home directory is "/usr/larry"; his login program is, by default, the shell program.

To assign a password you must exit the editor and use the "password" command. The "password" command enters the encrypted form of the password in the password file.

2.5 Deleting a User

Eventually, for one reason or another, you will need to remove a user from your system. First, you must determine if you want to save any of the files in that user's directories. If you do, copy them to another directory. Next, delete the unwanted files and directories from the system. It is not actually essential to sort through the files this way before removing the user from the system. However, it is wise to do so before much time has elapsed so that the content and purpose of the files are fresh in your mind.

The process of removing the user from the system is quite simple. All you need to do is edit the file "/etc/log/password" and delete the line which contains the relevant entry.

Chapter 3

Standard Devices

3.1 Introduction

Each version of the UniFLEX Operating System is capable of supporting a particular array of hardware devices. The number and kind of devices is system-dependent. Consult the table of standard devices that comes with the notes on hardware setup to determine which devices your system can support.

Some of these devices are "made" when you use the command "crdisk" to make a system disk. These devices are placed in the device directory, "/dev". You can see a list of these devices by executing the command

```
dir /dev
```

After creating a system disk you should delete from the device directory both the block and character references to any disks which are not physically connected to your system. Likewise, you should delete entries for any tape devices which are not connected to the system. If you fail to do so, a user who references one of these nonexistent devices may crash the system. Except for a tape device you do not need to delete the entry for a nonexistent character device that is not associated with a block device.

For instance, every version of the operating system can support four eight-inch floppy disk drives. The command "crdisk" creates two floppy disk devices. In order to be complete, every disk requires two entries in the directory "/dev": a block device and a character device. For the two floppy disk devices that are created by "crdisk", these devices are "fd0", "fdc0", "fd1", and "fdc1". If your system has only one floppy disk drive connected to it, you should delete the entries for the second drive with the following command:

```
kill /dev/fd1 /dev/fdc1
```

3.2 Adding a Device

If you do delete devices from your device directory and later obtain the necessary hardware, you can easily recreate the devices. A device is made by the "makdev" command. The syntax for this command is

makdev <dev_name> <dev_type> <major_dev_num> <minor_dev_num>

The "makdev" command creates a file named <dev_name> and stores the major and minor device numbers, which describe the device to the operating system, in the file descriptor node (fdn) for that file. You can put a device in any directory in which you have write permission. However, to ensure that all aspects of the operating system function smoothly, you should create them in the directory "/dev". With the exception of terminals (see Section 3.3), you can name a device almost anything you want. However, the last component of the name of the block device associated with a disk must have the following form:

<str_of_letters><str_of_digits>

The last component of the name of the character device associated with that disk must be identical to the name of the block device except that the letter 'c' must appear between the string of letters and the string of digits:

<str_of_letters>c<str_of_digits>

Usually, of course, it is best to give a device a name that is easy to remember.

The second argument, <dev_type>, designates the device as a block device, 'b', or a character device, 'c'. As mentioned earlier, all disks must have two devices: one block device and one character device. All other devices, including but not limited to terminals and printers, are character devices.

The major device number tells the operating system which set of device drivers to use for the device; the minor device number indicates which particular physical device to associate with <dev_name>. A list of standard devices and their corresponding major and minor device numbers is supplied with the UniFLEX Hardware Setup Instructions.

For example, suppose that you had deleted the entries for the second floppy drive ("fd1" and "fdc1") from the device directory and that you now want to add a second drive to your system. You must add both a block device and a character device to the device directory.

In current releases of the UniFLEX Operating System all block devices have the same major device number, 0. The first four (beginning with 0) minor device numbers for this major device number designate floppy disk drives 0 through 3. In all versions, the major device number for a character device associated with a floppy disk drive is 3. The first four minor device numbers for this major device number also designate floppy disk drives 0 through 3. Thus, the two commands necessary to create the floppy disk devices are

```
makdev /dev/fd1 b 0 1
makdev /dev/fdc1 c 3 1
```

You can see exactly how the operating system creates the devices on your system by looking at a listing of the "crdisk" command.

3.3 Adding a Terminal

Different versions of the UniFLEX Operating System can support different numbers of terminals. However, all versions of "crdisk" make devices for the maximum number of terminals. It is not necessary to delete the entries for unused terminals from the directory "/dev".

If for some reason you do delete the entry for a terminal and you want to recreate it, you can do so with the "makdev" command. The command is used for terminals just as it is used for other devices with the exception that the last component of the name of a terminal must be

```
tty<num>
```

where <num> is a two-digit number between 00 and the number of the last terminal. Terminal 00 must be the console--that is, the terminal from which you boot the system.

The operating system maintains a file, "/etc/ttylist", which contains information about all the terminals on the system. Each entry consists of a plus sign, '+', or a minus sign, '-', followed by a two-digit

number representing a particular terminal. A plus sign tells the operating system to activate the terminal by sending it a login prompt; a minus sign, to ignore it.

No matter how many terminals your system can support, the "crdisk" command makes only two active terminals: "tty00" and "tty01". If you add any other terminals to your system, you must edit the file "/etc/ttylist" and change the minus sign in the first column of the corresponding entry to a plus sign. If you remove a terminal from the system, change the plus sign to a minus sign. Depending on the particular arrangement of the wiring on your system, failure to do so may result in a degradation of system performance.

3.4 Adding a Serial Printer to a Terminal Port

All versions of the UniFLEX Operating System have two devices for serial printers in the directory "/dev". These devices use drivers that expect the printers to be connected to ports specifically reserved for them (see UniFLEX Hardware Setup Instructions).

You can put additional serial printers on your system by connecting them to terminal ports. You might also want to connect a printer to a terminal port if you have only one printer. By doing so you may be able to eliminate one serial interface card. In addition, if you are using the Enhanced Printer Spooler, you may have to connect your printer to a terminal port (see the documentation for the Enhanced Printer Spooler).

In any case, if you do connect a serial printer to a terminal port, you must make sure that a minus sign precedes the corresponding entry in the file "/etc/ttylist" (see Section 3.3)

Chapter 4

Important Directories

4.1 Introduction

The command which creates a system disk, "crdisk", creates several subdirectories in the root directory. Each subdirectory contains a set of related files. Grouping them in subdirectories helps to organize the various features of the operating system and to make them readily accessible. This chapter describes the kind of material found in each of the subdirectories in the root.

4.2 /act

The directory "act" contains files related to system accounting. When you boot the system, the operating system creates a file in "/act" called "utmp". When the system is in multi-user mode, the operating system writes the name and terminal number of each user who is using the system in this file. It also enters the time at which the user logged in. When the system enters single-user mode, the file is truncated to a length of 0.

The "who" command reads this file to obtain its output. If you ever forget which mode you are in, you can simply execute the "who" command. If the only response is a system prompt, no information is in the file "utmp", so the system is in single-user mode. Or, you can execute the following command:

```
dir +l /act
```

If the length of the file "utmp" is 0, the system is in single-user mode.

If a file named "history" exists in the directory "/act", the operating system uses it to maintain an account of the use of the system (see Section 1.9). Additional accounting software (available in the package User/System Accounting) also makes extensive use of this directory.

4.3 /bin

The name "bin" is short for binary files. This directory contains the most commonly used UniFLEX commands, such as "dir", "edit", and "kill".

4.4 /dev

The directory "/dev" contains the names of all the physical devices--such as disks, terminals, and printers--that are available on the system. As mentioned previously (see Section 3.1), you should delete from this directory both the block and character references to any disks which are not physically connected to the system. An accidental reference to one of these nonexistent devices may crash the system.

4.5 /etc

Except for the commands "login" and "print", the directory "/etc" contains commands which are intended for use only by the system manager. Such commands include "diskrepair", "crdisk", "makdev", and "shutup".

4.6 /gen

The directory "/gen" contains two directories: "help" and "errors". The directory "help" contains brief descriptions of the usage of most of the UniFLEX commands. The "help" command uses the information in this directory. The directory "errors" contains a file called "system" which is a binary listing of UniFLEX error numbers and messages. Various programs use this file to interpret any errors they receive from the operating system.

4.7 /lib

The directory "/lib" contains files which are sources containing definitions and defining structures that can be used by assembly language programs. If you add any compilers to your operating system, the run-time libraries for the compilers are placed in "/lib".

4.8 /lost+found

The directory "/lost+found" is for use by the utility "diskrepair" (see Chapter 7). If "diskrepair" finds any unreferenced files while it is checking the disk, it places them, by default, in "/lost+found". Although the files are no longer where they belong, the information they contain is preserved, as is the name of the owner.

4.9 /tmp

The directory "/tmp" is for use by programs which need to create temporary files--that is, files that you do not need after you execute the program. Some UniFLEX commands use this directory. Any programs you write may also use it. Thus, if the system happens to crash while any programs that use temporary files are running, the temporary files are all in one place and you can easily delete them all when you reboot the system. You can do so automatically by putting the command

```
kill /tmp/*
```

in the file "/etc/startup" (see Section 1.14).

4.10 /usr

The directory "/usr" originally contains two directories: "bin" and "spooler".

The directory "/usr/bin", like "/bin", contains binary files. These files are the less commonly used UniFLEX commands--such as "history", "free", and "info".

The empty directory "/usr/spooler" is the directory in which you create directories for each of the printers on your system (see Section 1.10.1.2).

4.11 /usr2

The empty directory "/usr2" provides a node for the mounting of a device on the system (see Section 1.8).

Chapter 5

Fatal System Errors

5.1 Introduction

The errors documented in this chapter are all fatal errors. They cleanly halt the system so that the disk remains intact. They send the error message to the console, tty00.

5.2 System Unable to Boot

The following errors may occur when you try to boot the system:

bad sys — Bad system.
Either the file "uniflex" has not been installed or the system disk is a copy of a copy of the master disk. You cannot boot a disk unless it is a direct copy of a master disk.

no swap — No swap space.
No swap space exists on the disk in the swap device. You must either reformat the disk with the 'r' option so that it contains some swap space or use another disk, which does contain swap space, as the swap device.

rter — Root error.
The boot program cannot read the root device. Use the "tune" command to verify that specification of the root device is correct (see Section 6.3.8). If it is incorrect, change it. If it is correct, the message is indicative of a failure on the disk in the root device.

stofl — System table overflow.
The tables that the operating system must construct are collectively larger than the space allowed for them. You can use the "tune" command to change the parameters that control the sizes of these tables--such as "buffers" and "locked_recs" (see Sections 6.3.1 and 6.3.5).

5.3 After Loading the Operating System

The following errors occur only after the file "/uniflex" is successfully loaded in memory:

fdngn Fdn gone.
This error is indicative of a hardware failure.

map No map.
This error is indicative of a hardware failure.

mntgn Mount gone.
This error is indicative of a hardware failure.

mtofl Memory table overflow.
This error is indicative of a hardware failure.

mtufl Memory table underflow.
This error is indicative of a hardware failure.

sirblk SIR block.
This error is indicative of a hardware failure.

swap er Swap error.
The system either cannot read from or write to the swap device. This error is indicative of a failure in the hardware or the medium. If the failure is in the medium, the "devcheck" command should locate the problem. Any bad blocks discovered by the utility can be placed in the bad-blocks file with the "badblocks" command. Note, however, that the swap space used for an individual task must be contiguous. Therefore, if defects occur in the swap space, you may need to allocate more swap space (which can only be done by reformatting the disk) in order to avoid running out of contiguous space.

swap ofl Swap overflow.
The table of internal system swap space overflowed. You cannot alter the size of this table. Contact Technical Systems Consultants if this error occurs.

swap space Swap space error.
The swap space is not large enough for the demand placed on it. You can increase the size of the swap space by reformatting the disk. Alternatively, if the swap device is the same as the root device, you can use the command "tune"

to change the swap device to another device which has more swap space (see Section 6.3.10).

syswp er — System swap error.
An error occurred in the management of the system's internal swap space. This error is indicative of a hardware failure.

tskofl — Task overflow.
No space remains in the internal task table. You can change the size of the task table with the command "/etc/tune" (see Section 6.3.11).

txtofl — Text overflow.
The users tried to execute too many shared-text programs simultaneously. You can change the number of shared-text programs allowed with the command "/etc/tune" (see Section 6.3.12).

xnam — Internal table overflow.
This error is indicative of a hardware failure.

Chapter 6

Fine-tuning the UniFLEX Operating System

6.1 Introduction

The "tune" command is used to alter certain parameters which govern the behavior and performance of the UniFLEX Operating System. Changing these parameters affects not only the behavior but also the performance of the operating system. Because different systems are used and stressed in different ways, the optimal settings for these parameters vary from site to site. Careful tuning of the operating system allows you to get the best performance from your particular system.

The "tune" command changes the values of the parameters in the file specified on the disk only, not in memory. Therefore, the changes have no effect until you boot the operating system from the modified version.

The syntax for the command is

```
/etc/tune <file_name> [<param_list>]
/etc/tune <file_name> [+r]
```

6.2 Modes of Operation

The "tune" command operates in three modes: read-only, interactive, and automatic. In read-only mode "tune" displays the current value, in the specified file, of each of the parameters it can alter. As system manager you can execute "tune" in read-only mode by specifying the 'r' option. (A user who does not have write permission in the file being tuned can only execute "tune" in read-only mode. In such a case the 'r' option is not necessary; "tune" automatically performs in read-only mode.)

If the user has write permission in the specified file and specifies neither a parameter list nor the 'r' option, "tune" executes in interactive mode. In this mode it displays current values one by one. To change the value of a parameter, enter the new value and a carriage return following the display. To leave the value as is, type just a carriage return.

The "tune" command imposes certain restrictions on the values of the parameters it alters. It also imposes a limit on the amount of memory collectively used by the following parameters: buffers, iolists, files, locked_recs, mounts, and tasks. If you enter a value in interactive mode which violates one of these restrictions, "tune" does not alter that value. Rather, it responds with an error message and waits for you to enter a new value.

Instead of going through the entire list of parameters interactively, you can specify on the command line which values "tune" is to change. The value of a parameter not mentioned on the command line does not change. After it makes the changes, "tune" displays a list of all parameters and the values you specified. If any of the values violates the restrictions mentioned previously, "tune" displays a message to that effect and sends a bell (control-G) to the terminal. Although the display shows the values you specified, "tune" does not make changes in the file if the changes violate the restrictions.

6.2.1 Arguments

The "tune" command takes one obligatory and one optional argument. These arguments are

<file_name>	The name of a file that contains a copy of the operating system.
<param_list>	An optional list of the parameters to change and of the values to assign to them.

6.2.2 Format for Arguments

The format of each element of the optional argument, <param_list>, is as follows:

<param_name>=<decimal_num>

Section 6.3 explains what parameters you can adjust.

6.2.3 Options Available

The "tune" command supports one option, the 'r' option, which tells "tune" to operate in read-only mode. The 'r' option is incompatible with the use of <param_list>. If you specify both, "tune" returns an error message.

6.3 Adjustable Parameters

The "tune" command can alter up to fourteen parameters. Table 6-1 briefly describes these parameters; they are discussed in more detail later in this section.

Table 6-1. Parameter Names and Descriptions

<param_name>	Description
buffers	Number of system buffers.
iolists	Maximum number of lists of I/O characters.
DST	Flag for the observation of Daylight Savings Time (0 indicates it is not observed locally; 1, that it is).
files	Maximum number of files that can be open at one time.
locked_recs	Maximum number of entries allowed in the table of locked files.
mounts	Maximum number of devices that can be mounted.
pipe_dev	Device number of the pipe device.
root_dev	Device number of the root device.
seek_rate	Seek rate of the floppy disk drive.
swap_dev	Device number of the swap device.
tasks	Maximum number of tasks the system can simultaneously execute.
text_segs	Maximum number of unique shared-text programs that the operating system can simultaneously execute.
time_zone	Time difference in minutes between local time and Universal Time. A positive value of "time_zone" indicates the number of minutes west of Greenwich; a negative value, the number of minutes east.
user_tasks	Maximum number of tasks allowed to each user.

The values for all these parameters are originally set when one of the "/uniflex" files is copied from the master disk to a system disk. These default values, as well as the limits imposed on each parameter, are

shown in Table 6-2.

Table 6-2. Default Values and Limits for "tune" Parameters

<param_name>	Default	Minimum	Maximum
buffers	sd	8	64
iolists	sd	16	255
DST	0	0	1
files	sd	16	255
locked_recs	32	0	Value of "files"
mounts	5	2	32
pipe_dev	sd	0	Last block device number
root_dev	sd	0	Last block device number
seek_rate	0	0	sd
swap_dev	sd	0	Last block device number
tasks	sd	8	64
text_segs	20	2	20
time_zone	300	-1440	1440
user_tasks	10	5	Value of "tasks"

Notes: sd = system-dependent

The object of changing these parameters is to make your version of the UniFLEX Operating System perform optimally on your particular system. Details of the parameters and the advantages and disadvantages of changing them are discussed in the following sections. You can determine the default values for your particular system by running the "tune" command in read-only mode on the appropriate "uniflex" file on your master disk.

6.3.1 System Buffers

The parameter "buffers" refers to the number of blocks reserved for the buffer cache. Normally, the buffer cache contains 32 blocks; the limits are 8 and 64 blocks. The number of blocks reserved for the buffer cache must be a multiple of 8.

When the operating system searches the buffer cache for a particular block, it must search sequentially. Thus, the larger the buffer cache, the greater the time spent searching it. Depending on how heavily your system is used, you may see a degradation in system response as you increase the size of the cache.

The addition of blocks to the buffer cache also results in an increase in the size of the operating system and, consequently, a decrease in the amount of memory available to users. Each block added to the cache uses approximately 4K of memory.

The advantage of increasing the size of the buffer cache is, of course, that the system does not need to access the disk as often. The overall speed of operation may, therefore, increase.

6.3.2 Lists of I/O Characters

The operating system buffers both input to and output from terminals. The parameter "iolists" refers to the maximum number of buffers of I/O characters the system can support. The number of I/O buffers supplied with the operating system is system-dependent. The minimum is 16; the maximum, 255. Each buffer occupies 32 bytes of memory.

A single terminal can access no more than 20 buffers. The maximum useful value for "iolists" is, therefore, the number of terminals on the system multiplied by 20. All systems, however, are limited to the maximum of 255. If your system slows down noticeably when terminal activity is high, you may want to increase the value of "iolists". The only cost is the 32 bytes of memory used by each buffer.

6.3.3 Daylight Savings Time

The Daylight Savings Time flag indicates whether or not Daylight Savings Time is observed locally. A value of 0 indicates that it is not; a value of 1, that it is. The default value is 0. The operating system assumes that Daylight Savings Time begins and ends according to the dates used in the United States--the last Sunday in April and the last Sunday in October. Setting the flag for Daylight Savings Time does not affect the performance of the operating system.

6.3.4 Number of Open Files

The parameter "files" refers to the number of files that can simultaneously be open on the system as a whole. The default value is system-dependent. The minimum value is 16; the maximum, 128.

The system needs to store approximately 100 bytes of information about each file that is open. This information includes details about the file itself as well as control information such as what device the file is located on. Thus, each additional file requires approximately 100 bytes of memory and decreases the memory available to users accordingly.

The operating system maintains a table that tells which files are open. As the number of entries in the table increases, the time required to search the table also increases. You may, therefore, see some degradation in system response as you increase the value of "files".

On the other hand, you may see an improvement in performance as you increase the value of "files". The information about an open file is stored in memory. Even when a user closes the file, the information about the file stays in memory until this cache of file descriptor nodes (the fdn cache) is full. When a user opens a file, the operating system first looks in the fdn cache for the information it needs about the file. If the information is already there, the time that would be required to access the disk is saved. Thus, as the size of the fdn cache increases, the speed of the operating system may increase.

The point at which the increase and decrease in system performance balance each other depends on the way in which a particular system is used.

Although you can change the number of files that can simultaneously be open on the system by changing the value of "files", you cannot change the number of files that one task can open. That number is restricted to 16.

6.3.5 Number of Locked Files

Whenever the system call "lrec" is used to lock a file, the operating system makes an entry in its table of locked files so that other users can check to see whether or not the files they want to access are

locked. The parameter "locked_recs" determines how many entries this table can contain. If you try to lock a file when the table is full, you receive a lock error from the operating system. Individual tasks may handle this error differently, but the net result is that because the operating system cannot lock the file until there is room in the table, the user has to wait.

The cost of increasing the value of "locked_recs" is slight. If the users on your system lock files frequently, you may want to increase the value of "locked_recs" above the default value of 32. The minimum value you can use for this parameter is 0; the theoretical maximum, 128. However, for any given system the practical maximum is the same as the value of "files".

6.3.6 Number of Mounted Devices

The value of the parameter "mounts" determines how many devices can simultaneously be mounted on the system. The default value for "mounts" is 5. The minimum value is 2; the maximum, 32. Although you cannot unmount the root directory, it is mounted and, therefore, counts as a mounted device.

Each device you add by increasing the value of "mounts" uses approximately 540 bytes of memory. The effect on the speed of the operating system is negligible.

6.3.7 Pipe Device

The value of the parameter "pipe_dev" describes which block device the operating system is to use for creating pipes. It stores the number as a 2-byte hexadecimal number whose first byte is the major device number of the pipe device and whose second byte is the minor device number of the pipe device. To specify the value in decimal (as you must when using "tune"), use the following formula:

<pipe_dev> = 256*<major_dev_num> + <minor_dev_num>

Block devices in current implementations of the UniFLEX Operating System all have the same major device number, 0.

The default value for "pipe_dev" is system dependent. The minimum value you can use for this parameter is 0; the theoretical maximum, 32. However, for any given system the practical maximum is the the value of the largest minor device number. Changing the pipe device does not affect the performance of the operating system.

The pipe device must always be the same as either the root device or the swap device. The "tune" command does not check to see if you have conformed to this restriction. It is, therefore, possible to specify a pipe device that matches neither the root device nor the swap device. In such a case, you can still boot that version of the operating system, but when you try to use a pipe, the system crashes.

6.3.8 Root Device

The parameter "root_dev" tells the operating system which device contains the root directory. The value of "root_dev" is calculated in the same way as the value of "pipe_dev" (see Section 6.3.7). The default, minimum, and maximum values are also the same. Changing the root device does not affect the performance of the operating system.

6.3.9 Seek Rate of the Floppy Disk Drives

The floppy disk driver tells the hardware how fast to try to move the heads of the floppy disk drives from one track to an adjacent track. The parameter "seek_rate" specifies this rate. If the rate is too fast, the system does not function properly. If it is too slow, the system wastes time.

The default value of "seek_rate" is 0. The minimum is also 0; the maximum is system-dependent, but the largest value for any system is 3. In current implementations of the UniFLEX Operating System, these values correspond to actual seek rates as follows:

<seek_rate>	Actual Seek Rate
0	3 msec
1	6 msec
2	10 msec
3	15 msec

The values for future implementations of the operating system may differ. A value of 0 for <seek_rate>, however, will always represent the fastest seek rate.

6.3.10 Swap Device

The parameter "swap_dev" tells the operating system which device to use for swapping. The value of "swap_dev" is calculated in the same way as the value of "pipe_dev" (see Section 6.3.7). The default, minimum, and maximum values are also the same.

Changing the swap device to a device that is exclusively used for swapping may improve system performance because the head of that disk drive always remains over the swap space.

6.3.11 Number of Tasks Supported

Although only one task can actually occupy the CPU at any given time, the operating system can support more than one "active task". An active task is simply a task to which the operating system has assigned a task ID. The operating system maintains a table of active tasks. The parameter "tasks" determines how many entries this table can hold. The default value of the number of tasks supported is system-dependent. The minimum number is 8; the maximum, 64.

Each addition to the number of tasks the system can support requires approximately 30 bytes of memory. As the number of tasks that can be supported increases, the system may slow down.

The parameter "tasks" determines the number of tasks that the system can support as a whole. The number of active tasks that an individual user can have is determined by a different parameter, "user_tasks".

6.3.12 Shared-text Programs

Certain UniFLEX programs that are both moderately large and frequently used—such as the shell program, the editor, and BASIC—are treated as shared-text programs. No matter how many people are using these programs, the operating system needs only one copy of the program in memory. The parameter "text_segs" determines how many shared-text programs the operating system can support. The default value is 20. The minimum is 2; the maximum, 20.

For every increase in the value of "text_segs", the operating system must use approximately 20 to 30 bytes of memory. The system runs marginally faster as the value of "text_segs" decreases.

6.3.13 Setting the Time Zone

When you use the "date" command to set the date and time, the operating system converts the time you enter into the number of seconds that have passed since January 1, 1980, at the zeroth meridian (in Greenwich, England). As it makes this conversion it must adjust the result for the local time zone and for Daylight Savings Time, if it is in effect. The value of the parameter "time_zone" is the number of minutes difference between local time and Greenwich Mean Time (Universal Time). The value must represent a nonfractional number of hours—that is, it must be a multiple of 60. A positive value of "time_zone" indicates the number of minutes west of Greenwich; a negative value, the number of minutes east.

The default value of "time_zone" is 300 (the correct value for Layfayette, Indiana, the birthplace of Technical Systems Consultants). The minimum value is -1440; the maximum, 1440.

6.3.14 Number of Tasks per User

The parameter "user_tasks" determines the maximum number of tasks an individual user can simultaneously execute. The default value is 10. The minimum value you can use for this parameter is 5; the theoretical maximum, 25. However, for any given system the practical maximum is the same as the value of "tasks". The same number applies to all users, excluding the system manager.

The cost of increasing the value of "user_tasks" is negligible. You may want to lower the number of tasks allowed to each user if your system seems to be saturated.

Although varying the parameter "user_tasks" allows you to vary the number of tasks each user can simultaneously execute, you cannot alter the fact that any one shell program supports a maximum of five background tasks.

6.4 Examples

The following examples illustrate some uses of the "tune" command.

1. /etc/tune /uniflex +r
2. /etc/tune /usr2/uniflex2 tasks=32 swap_dev=1

The first example displays a list of the items that "tune" can adjust. The current value of each item in the file "uniflex" in the root directory appears in parentheses.

The second example changes the specified parameters in the file "uniflex2" (the mini-Winchester version of the operating system) in the directory "/usr2". Presumably, a system disk is mounted on "/usr2". This command sets the maximum number of tasks allowed on the system to 32 and defines floppy drive 1 as the swap device. In order for this particular version of the operating system to be able to perform swapping, a floppy disk formatted with swap space must be in floppy drive 1. The disk should not be mounted.

6.5 Error Messages

This section describes the error messages returned by "tune".

'r' option incompatible with command-line parameters.
The 'r' option, which tells "tune" to operate in read-only mode, conflicts with the specification of parameters on the command line. The command is aborted.

***Value out of range [num_1, num_2]
The value specified for a parameter is not within the range of acceptable values. The limits of the range are shown inside the square brackets.

***Value must be a multiple of <num>.
The value for the number of buffers in the system must be a multiple of 8. The value for the time zone must be a multiple of 60.

***Total system table space exceeded.
The specified change violates the restriction on the amount of memory collectively allowed to buffers, iolists, files, locked_recs, mounts, and tasks.

Chapter 7

Repairing a Damaged Disk

7.1 Introduction

The UniFLEX Operating System includes a utility, "diskrepair", which checks the structure of the disk or disks specified on the command line. The structure of a disk refers to the layout of and the connections among files, directories, free space, swap space, and other information that makes up the file system. "Diskrepair" detects any inconsistencies in the structure of the disk and, optionally, repairs them. Although "diskrepair" does not methodically search for and repair media (I/O) errors, it can take care of any bad blocks it encounters. If the 'a' option is in effect when "diskrepair" encounters an I/O error, it calls the utility "/etc/badblocks", which places the offending block in the bad-blocks file, "/.badblocks" (see the documentation of the "badblocks" command in the operating system manual).

While it is operating, "diskrepair" calls two other utilities--"blockcheck" and "fdncheck", which are both located in the directory "/etc". "Blockcheck" is concerned with the allocation of blocks on the disk. It locates problems such as duplicate blocks, missing blocks, and invalid block addresses. "Fdncheck" is concerned with the directories on the disk. It locates problems such as unreferenced files, directory entries with invalid associated files, and so forth. These errors are discussed in more detail later in this chapter.

"Diskrepair" performs some tasks that are not directly related to the logical structure on the disk. These preliminary tasks are essential to the proper performance of the utility. The heart of the program, which actually checks the structure of the disk, consists of the following six phases:

Phase 1--Check allocated blocks
Phase 2--Scan directories
Phase 3--Check unreferenced directories
Phase 4--Check file and directory links
Phase 5--Check free list
Phase 6--Check SIR information

This chapter discusses each of the phases, as well as the other tasks performed by "diskrepair", in chronological order. The appropriate error messages, with the exception of the messages resulting from physical errors, are documented with each section. Messages resulting from physical errors are documented in Section 7.14. In addition, Section 7.15 consists of an index of error messages which directs you to the page on which each message is explained.

7.1.1 Structure of a UniFLEX Disk

Before you can understand what "diskrepair" does, you must understand some things about the structure of a UniFLEX disk. When the operating system formats a disk, it writes a boot program into the first logical block on the disk, block 0. The block containing the boot program must always be in the same place so that the bootstrap loader in read-only memory (ROM) can find it. The second block, block 1, is the system information record (SIR). This block contains information describing the layout of the remainder of the disk. This information is essential to the successful execution of the operating system. The rest of the disk consists of swap space, volume space, and file descriptor nodes.

Swap space is the part of the disk reserved for the storage of tasks that are swapped out of main memory onto the disk. Swap space is not necessary on a disk that is to contain only data, but every system disk needs some swap space (see Section 1.5).

Normally, when a disk is formatted, most blocks on it are available for storing data in files. The addresses of available blocks are maintained in a "free list". When a file needs a block, the operating system removes the address of a block from the free list and associates that block with the file in question. The combination of the blocks used in files and the blocks in the free list is known as the volume space.

When you format a disk (using one of the versions of the "format" command), the operating system reserves a certain number of blocks (determined by the 'f' option) for file descriptor nodes (fdns). An fdn contains all the information that the operating system needs to know about a file. This information includes, but is not limited to, the type of file, the owner of the file, the size of the file, and the addresses of all the blocks that are a part of the file.

Whenever the operating system creates a file, it makes an entry in the parent directory. The entry contains the name of the file and the number of the fdn assigned to that file. It is possible for more than one directory entry to point to the same fdn; each of these entries is called a link. Each link results in another name for a file which already exists. However, no matter how many links there are to a file, only one fdn describes the file itself. Thus, each file on the disk should correspond to exactly one fdn.

The information in the SIR and the fdns establishes the logical structure of the disk. "Diskrepair" checks this structure and, optionally, repairs the errors it finds. It is able to do so because a certain amount of redundancy exists on the disk. For instance, the link count for a file (the number of directory entries that point to the fdn for that file) is stored in the fdn itself for quick reference. However, as "diskrepair" looks at the structure of the disk, it checks every directory entry and keeps track of how many times each fdn is referenced. If the number of direct references (or links) to an fdn does not agree with the the number stored in the fdn itself, it is a simple matter for "diskrepair" to change the number in the fdn.

7.1.2 Physical Errors on the Disk

"Diskrepair" is not a substitute for maintaining proper backups. For one thing, it cannot repair physical media errors. A physical error is an error from the hardware, usually caused by a bad spot on the medium, which prevents the operating system from reading from or writing to it. If at any time "diskrepair" encounters a physical error on the disk, it prints one of the following messages:

```
Error reading block <block_num>.
Error writing block <block_num>.
Error reading fdn <fdn_num> in block <block_num>.
Error writing fdn <fdn_num> in block <block_num>.
```

followed by the prompt:

```
-----> Continue?
```

If you see one of these messages, your disk is probably physically damaged. In general, if you choose to continue with "diskrepair", the results are entirely unpredictable. They depend on precisely which block is damaged. Continuing with "diskrepair" may cause further damage

to the disk, but in some cases, it may be the desired course of action. If you choose not to continue, "diskrepair" aborts.

We suggest that you respond negatively to the prompt to continue the first time "diskrepair" reports an I/O error and that you immediately rerun "diskrepair". It is possible--though unlikely--that the I/O error is a soft one and will not recur. If the error does recur, respond negatively to the prompt to continue and immediately rerun "diskrepair" with the 'a' option.

In many cases if you choose to continue, you receive another message which describes what "diskrepair" was trying to do when it encountered the I/O error (see Section 7.14).

7.1.3 Limitations of "diskrepair"

"Diskrepair" cannot solve all the problems your disk may have. As mentioned in the preceding section, it cannot fix physical media problems (but see Section 7.2.1). As for problems with the logical structure of the disk, "diskrepair" can only repair an error if the damaged information is redundant--that is, if there is some way of determining what the information should be. It cannot, for example, repair a badly damaged SIR; nor can it repair a disk if the root directory is severely damaged.

Now that you have been warned that "diskrepair" cannot fix all the problems that may arise when a disk is damaged, let's look at how it functions and at the large number of things that it can do.

7.2 The Command Line

The syntax for "diskrepair" is as follows:

```
/etc/diskrepair <dev_name> [<dev_name_list>] [+<abfimnpqruvz>]
```

where <dev_name> is the name of a device to check. A brief description of the options which are available follows:

a Automatically place in the file "/.badblocks" any bad blocks encountered, and continue to run "diskrepair" until the disk is repaired or the program has executed ten times.
b Perform "blockcheck" only.
f Perform "fdncheck" only.
i Ignore the restriction on repairing the disk in multi-user mode. This option should be used if and only if the system cannot operate in single-user mode.
m Ignore missing blocks.
n Do not attempt to fix errors.
p Prompt for permission to repair.
q Use quiet mode.
r Rebuild the free list whether or not it is in error.
u Report on the usage of disk blocks.
v Use verbose mode.
z Do not print messages about possible errors in the sizes of files.

If you execute "diskrepair" without any options, it does what it can to repair structural errors on your disk. You can, however, modify its behavior by specifying various options on the command line. In particular, if you exercise the 'p' and 'v' options, "diskrepair" reports its progress in greater detail and prompts you for permission before making any repairs. In order to give you a more detailed explanation of the "diskrepair" command, this chapter assumes that you have specified both the 'p' and 'v' options on the command line. In the absence of the 'p' option, "diskrepair" behaves as if it had prompted you for a response before each repair and you had answered positively.

Detailed descriptions of the options follow.

7.2.1 The 'a' Option

The 'a' option automatically places any bad blocks found by "diskrepair" in the file "/.badblocks". It also runs "diskrepair" continuously until either the disk is fixed or the program has executed ten times.

7.2.2 The 'b' Option

The 'b' option instructs "diskrepair" to run only the "blockcheck" portion of the utility. This procedure is often considerably faster, but still provides a fairly complete assessment of the validity of the structure of the disk.

7.2.3 The 'f' Option

The 'f' option instructs "diskrepair" to run only the "fdncheck" portion of the utility. This option is useful if you suspect a problem exists in the directory structure, but the result is by no means a thorough check of the structure of the disk.

7.2.4 The 'i' Option

By default, "diskrepair" does not allow the user to repair the root device (system disk) unless the system is in single-user mode. However, certain systems--such as the Tektronix 4404--cannot operate in single-user mode. In order to repair the system disk on such a system the user must specify the 'i' option, which tells "diskrepair" to ignore this restriction. Under no circumstances should this option be used on a system which can operate in single-user mode.

7.2.5 The 'm' Option

The operating system maintains a list of blocks available for use called the free list. A missing block is any block in the volume space which is not a part of any file and is not in the free list. The existence of such blocks is a harmless error in the structure of the disk. "Diskrepair" generally places missing blocks in the free list. The 'm' option, however, instructs "diskrepair" not to rebuild the free list solely on account of missing blocks. This option reduces the time required for "diskrepair" to run if missing blocks are the only problem in the free list.

7.2.6 The 'n' Option

The 'n' option tells "diskrepair" to report all errors but to make no attempt to fix them. Therefore, "diskrepair" opens the device for reading only. This option is useful for checking the structure of a disk without risking the loss of data during repairs.

7.2.7 The 'p' Option

If you specify the 'p' option, "diskrepair" reports each error, followed by a prompt requesting permission for the proposed repair. All prompts require an answer of either 'y' ("yes") or 'n' ("no").

Many repairs result in the loss of data. (You can generally infer what has been lost from the messages "diskrepair" displays.) Judicious use of the 'n' and 'p' options not only allows you to assess the damage to the disk and decide which information you are willing to sacrifice during the repair process but also gives you the opportunity to try to salvage the data before repairing the disk. Methods of salvaging data from a damaged disk are discussed in Chapter 8.

7.2.8 The 'q' Option

The 'q' option suppresses certain warnings and messages from "diskrepair". Several conditions exist which, while not technically errors in the structure of the disk, may cause problems. These conditions usually result in a warning message; the 'q' option suppresses such messages.

7.2.9 The 'r' Option

By default, if "diskrepair" finds that the free list is in error, it rebuilds it. The 'r' option instructs "diskrepair" to rebuild the free list whether or not it contains errors. This option may save some time if you know that the free list is bad. You can also use it to reduce fragmentation within the free list.

7.2.10 The 'u' Option

The 'u' option generates a report on the block usage of the specified device. This report is printed at the end of the "diskrepair" operation. It contains statistics on (1) the number of each type of file in the file system and the total number of files in the system, (2) the number of unused blocks and the number of used blocks, including a breakdown of how the used blocks are allocated, and (3) the number of free fdns and the number of fdns in use.

7.2.11 The 'v' Option

"Diskrepair" operates in one of two modes: simple or verbose. Simple mode is selected by default; verbose mode is selected by the 'v' option. In simple mode "diskrepair" reports only those errors which require the deletion of either files or directory entries. In verbose mode all errors are reported. In addition, informative messages are printed describing what phase "diskrepair" is performing.

In verbose mode the 'p' option causes "diskrepair" to prompt for permission regarding all errors. In simple mode the user is prompted only for those errors which require the deletion of either files or directory entries; all other errors are automatically repaired without prompting.

7.2.12 The 'z' Option

Normally, "diskrepair" reports a possible error in the size of a file. The 'z' option suppresses such messages. If the 'q' option is in effect, the 'z' option is redundant.

7.3 Preliminary Checks

Before "diskrepair" even begins to check the structure of the disk, it makes several preliminary checks, which are necessary to ensure that the utility can function properly.

7.3.1 Command-line Options

First of all, "diskrepair" checks the validity of each character specified as an option on the command line. If any of these characters is not a valid option, "diskrepair" tells you

```
Unknown option: <char>
```

If the options you specify conflict with each other, it tells you

```
Conflicting options.
```

In either case, "diskrepair" aborts.

7.3.2 Specified Device

Next, "diskrepair" looks at the device or devices you specified on the command line. If you specify a nonexistent device or if you fail to specify a device, "diskrepair" responds with the appropriate message:

```
No such device.
"<dev_name>" ignored.
```

or

```
No device specified.
```

The first of these messages is not fatal unless the user specified only one device on the command line. The second is always fatal.

"Diskrepair" can only operate on a block device. Therefore, it must determine whether or not the device specified on the command line is a block device. If it is not, "diskrepair" issues the following message:

```
Not a block device.
```

This message, too, is fatal to "diskrepair".

7.3.3 Permissions

If you execute "diskrepair" without the 'n' option, you must have both read and write permission on the specified device; with the 'n' option, you need only read permission. If you do not have the necessary permissions, "diskrepair" informs you:

```
Permission denied.
```

The program then aborts.

7.3.4 Checking the Mode of Operation

By default, "diskrepair" cannot repair the root device unless the system is in single-user mode (but see Section 7.2.4). Therefore, if the device specified on the command line is the root device, "diskrepair" must determine what mode the system is operating in. If for some reason it cannot access the file "/act/utmp" (it is the length of this file that tells what mode is in effect--see Section 4.2), you receive the following message before "diskrepair" aborts:

```
Can't determine mode.
```

If "diskrepair" finds that the system is in multi-user mode, it reports

```
Must be in single-user mode.
"<dev_name>" ignored.
```

"Diskrepair" ignores the system disk and proceeds with the next argument on the command line--if one exists.

7.3.5 Unmounting a Mounted Disk

Any alterations that "diskrepair" makes must be made when the disk is not in use. Therefore, "diskrepair" determines whether or not the specified disk is mounted, and, unless you specify the 'n' option, it unmounts a mounted disk before proceeding. This procedure ensures that no user can access the disk while "diskrepair" is repairing it. If any user's working directory is on the device or if any file on the device

is being accessed when "diskrepair" tries to unmount it, the unmount procedure fails and the following message appears on your screen:

```
Device is busy.
```

If "diskrepair" encounters some other problem when it tries to unmount the device, it responds

```
Unmount error:  <error_num>
```

where <error_num> is the number of the UniFLEX error that caused the failure. Consult the operating system manual for an explanation of the error.

If the 'n' option is in effect, "diskrepair" does not need to unmount a mounted disk because it will not have to write to the disk. However, when running "diskrepair" with the 'n' option, you should make sure that no one else is using the disk that is being tested. If you run "diskrepair" while the disk is in use, the results are unreliable.

7.3.6 Status of the Root Directory

Finally, "diskrepair" tries to read the fdn which describes the root directory. If, for any reason, it cannot access this fdn, it reports

```
Can't stat root.
```

This message means that "diskrepair" cannot get the status of (read the fdn for) the root directory. This error is fatal to "diskrepair". You may be able to salvage some of the data on the disk (see Chapter 8), but you must reformat the disk.

7.4 Calling "blockcheck"

At this point, "diskrepair" calls the utility "/etc/blockcheck". If it

cannot read or execute that file, it tells you:

```
Can't call /etc/blockcheck.
```

It then aborts.

After successfully accessing "blockcheck", "diskrepair" checks to make sure that it is the proper version of the utility. If it is not, it aborts after reporting:

```
/etc/blockcheck is invalid.
```

7.4.1 Abnormal Termination of "blockcheck"

If for any reason "blockcheck" terminates abnormally--that is, receives a program interrupt from the operating system--"diskrepair" issues the following message before it aborts:

```
Blockcheck terminated abnormally (status = <num>).
Diskrepair aborted for "<dev_name>".
```

Such a message is not indicative of a problem with either "diskrepair" or the device. You should try to run "diskrepair" again, for the problem may not recur.

7.4.2 Improper I/O Redirection

When testing the structure of a disk, it is impractical to try to redirect the output (the results of the test) to a file on the disk you are testing. If you do try to do so, you receive the following message:

```
Output directed to device under test.
```

In such a case, "diskrepair" aborts.

While it is checking the validity of any redirected output, "diskrepair" must read the fdn of whatever file is open as standard output. If, for any reason, "diskrepair" cannot read this fdn, it reports

Can't stat std. output.

In such a case you should reexecute "diskrepair" with the terminal as standard output.

7.5 Preliminary Checks on the SIR

7.5.1 Accessing the SIR

Before proceeding with the tests on the structural integrity of the disk, "blockcheck" tries to read the SIR. If the SIR has been damaged so badly that "blockcheck" cannot read it, it reports:

Can't read System Information Record.

This error is fatal to "diskrepair". You may be able to salvage some information from the disk (see Chapter 8), but you must reformat it.

7.5.2 Size of Disk

"Diskrepair" checks to see that the size of the disk is within the range that it can handle. The current limit is approximately 160 Megabytes. If the data in the SIR indicate that the disk is larger than this limit, "diskrepair" issues the following message:

Disk too large or bad size in SIR.
"<dev_name>" ignored.

This error, too, is fatal to "diskrepair". You may be able to salvage some information from the disk (Chapter 8), but you must reformat it.

7.5.3 Fdn Count

"Diskrepair" reads the SIR to determine how many blocks on the disk are reserved for use as fdns. It checks to see that (1) the number of fdns does not exceed 65,535 (the maximum allowed by the operating system) and (2) the number of blocks allocated for fdns does not exceed the size of the disk. In either case, "diskrepair" issues the following message:

```
SIR fdn block count error: <num_of_fdns>
"<dev_name>" ignored.
```

This error is fatal to "diskrepair". You may be able to salvage some information from the disk (see Chapter 8), but you must reformat it.

7.5.4 First Block of Swap Space

The SIR contains the address of the first block in the swap space. It also contains information (the total number of blocks outside the swap space) that allows "diskrepair" to calculate independently what that address should be. If the calculated address is less than the address stored in the SIR, a contradiction exists: some blocks that are in the volume space are also in the swap space. If this situation arises, "diskrepair" issues the following message:

```
Data overlaps swap space.
------> Assume data correct and fix?
```

If you respond 'n', "diskrepair" aborts with the message

```
"<dev_name>" ignored.
```

If you respond 'y', "diskrepair" assumes that the address it calculated from the number of blocks outside the swap space is correct and rewrites the SIR with the calculated address as the address of the first block of the swap space.

If the calculated address is greater than the address stored in the SIR, the net result is a hole in the map of the disk: the volume space ends before the swap space begins. The intermediate blocks are inaccessible; they are allocated neither to the volume space nor to the swap space.

Although this situation wastes some disk space, it does not cause any logical inconsistencies in the structure of the disk. Therefore, "diskrepair" considers the disk intact.

7.6 The File "/.badblocks"

You can effectively hide blocks that are known to be bad by putting them in the bad-blocks file, "/.badblocks", with either the "format" or "badblocks" command (see Section 1.6). The operating system knows not to allocate any of the blocks which are in this file. "Diskrepair", too, must be aware of the presence of this file, so that it does not inadvertently put the bad blocks into the free list. Therefore, the last thing it tries to do before it starts to check the structure of the disk is to read the file "/.badblocks".

7.6.1 Accessing the Bad-blocks File

If "diskrepair" encounters an I/O error while trying to read the bad-blocks file, it reports the following error and continues:

```
Error checking ".badblocks" file:  Ignored.
```

Although "diskrepair" continues to run, it does not know about the blocks in the bad-blocks file. In such a situation the results are unpredictable. If problems arise, you may be able to salvage some of your data, (see Chapter 8), but you must eventually reformat the disk.

7.6.2 Validating the Bad-blocks File

If "diskrepair" can read the file but finds that (1) it is not a regular file or (2) the first block of the file is located where the boot sector, the SIR, or the root directory should be, it issues the following message and tries to continue:

```
Bad ".badblocks" file:  Ignored.
```

Since "diskrepair" cannot read the file "/.badblocks", it does not know which blocks are bad. It might, therefore, try to access a bad block. If it does, it informs you of a physical error. This error is also fatal to the "badblocks" command. In such a case you should salvage as much information as possible from the disk (see Section 8.3), but you must eventually reformat it.

7.7 Phase 1--Check Allocated Blocks

During phase 1 "diskrepair" reads every fdn on the disk. From reading the fdns it can determine which ones are inactive (do not describe a file), which ones describe block devices or character devices, and which ones describe regular files or directories. If an fdn refers to either a regular file or a directory, "diskrepair" continues reading the fdn to see how many blocks and which blocks are allocated to that file.

7.7.1 File Size

Once "diskrepair" knows how many blocks are allocated to a particular file, it compares that number to the size that is written in the fdn. Since the size is measured in bytes, this comparison can only determine if the size and the number of allocated blocks are compatible. For instance, if two blocks are allocated to the fdn, you would normally expect the file to contain between 513 and 1,024 bytes. If the size is not in this range, "diskrepair" reports

WARNING: Possible file size error in fdn <fdn_num>

"Diskrepair" does not attempt to correct such a discrepancy because it is possible for the numbers to disagree and to be correct. Only by considering the contents of the file in question can you be sure whether or not the discrepancy is an error. If the file was written using the text editor, this kind of discrepancy should not arise. If, on the other hand, the file was written by a program which used the system call "seek" to write to the file in a nonsequential manner, a discrepancy in the two numbers is possible. If the warning is indicative of a genuine error, you can fix it by copying or deleting the file.

7.7.2 Out-of-Range Blocks in Fdns

While "diskrepair" is reading fdns to determine which blocks are allocated to which files, it checks to make sure that the address of each block is a valid one. The addresses should correspond to blocks within the volume space on the disk. If an address which appears in an fdn corresponds to a block in the swap space, to a block in the fdn blocks, to the boot sector, to the SIR, or to a nonexistent block, "diskrepair" reports

Out-of-range block in fdn <fdn_num>.

It does not, at this point, do anything to correct the problem (see Section 7.9.4). The problem will be corrected later; however, the only way to fix the structure of the disk in such a case is to delete the file containing the out-of-range block. Before doing so, however, you may be able to salvage most of the file (see Section 8.3.2).

"Diskrepair" maintains a list containing the addresses of the out-of-range blocks it encounters. Currently, the maximum length of this list is twenty blocks. If "diskrepair" encounters more than twenty out-of-range blocks, it continues to report each one, but it tells you

Too many out-of-range blocks.

This message means that "diskrepair" cannot add any more blocks to the list. Of course, under such conditions it cannot completely repair the disk. Therefore, when "diskrepair" finishes, it tells you to run "diskrepair" again.

7.7.3 Blocks Duplicated in Fdns

A given block from the volume space should be allocated either to the free list or to exactly one fdn (hence, one file). If "diskrepair" finds that a block is allocated to more than one fdn, it keeps track of the number of the fdn in which the second, and any subsequent, allocations occur. If it does find any blocks that are duplicated in fdns, it enters phase 1B, in which it scans the fdns again in order to determine which fdn first claimed a duplicate block. It then prints a message for each fdn which contains a duplicate block. The message has the following form:

Duplicate block <block_num> in fdn <fdn_num>.

It does not, at this point, do anything to correct the problem (see Section 7.9.5). The problem will be corrected later; however, the only way to fix the structure of the disk is to delete all but one of the files that contain the duplicate block. Before doing so, however, you should be able to salvage most of the data in the files (see Section 8.3.3).

"Diskrepair" maintains a list containing the addresses of the duplicate blocks it encounters. Currently, the maximum length of this list is thirty blocks. If "diskrepair" encounters more than thirty duplicate blocks, it continues to report each one but also includes a message saying

Too many duplicate blocks.

This message means that "diskrepair" cannot add any more blocks to the list. Of course, under such conditions it cannot completely repair the disk, and when it finishes, it tells you to run "diskrepair" again.

7.8 Transition between Phase 1 and Phase 2

During the transition from phase 1 to phase 2 "diskrepair" calls the utility "fdncheck", which does some preliminary checking of its own before proceeding with the evaluation of the logical structure of the disk.

7.8.1 Calling "fdncheck"

At this point, "diskrepair" calls the utility "/etc/fdncheck". If it cannot read or execute that file, it tells you:

Can't call /etc/fdncheck.

It then aborts.

After successfully accessing "fdncheck", "diskrepair" checks to make sure that it is the proper version of the utility. If it is not, it aborts after reporting:

```
/etc/fdncheck is invalid.
```

7.8.2 Abnormal Termination of "fdncheck"

If for any reason "fdncheck" terminates abnormally--that is, receives a program interrupt from the operating system--"diskrepair" issues the following message before it aborts:

```
Fdncheck terminated abnormally (status = <num>).
Diskrepair aborted for "<dev_name>".
```

Such a message is not indicative of a problem with either "diskrepair" or the device. You should try to run "diskrepair" again, for the problem may not recur.

7.8.3 The File "/.badblocks"

The utility "fdncheck", like "blockcheck", tries to read the file ".badblocks" in the root directory. It is possible for "fdncheck" to have trouble reading the bad-blocks file even if "blockcheck" does not. If it does encounter a problem, it returns one of the error messages described in Section 7.6.

7.8.4 Reading the Root Directory

Before entering the next phase, "fdncheck" tries to read the root directory. If the root is so badly damaged that it cannot be read, "diskrepair" issues the following message:

```
Can't read root directory.
```

If the length of the root directory has been truncated to 0, you receive a message to that effect:

```
Root directory zero length.
```

These messages are both fatal to "diskrepair". In either case the disk cannot be repaired. Such a disk cannot be mounted; therefore, you cannot even attempt to salvage the information on it.

If "fdncheck" tries to read the root directory and discovers that the root fdn describes it as a regular file rather than a directory, it reports

```
Root fdn is not a directory.
------> Force into directory?
```

A negative response causes "diskrepair" to abort. Depending on the extent of the damage, you may be able to salvage some of the information on the disk (see Chapter 8). A positive response causes "diskrepair" to force the fdn to describe a directory and to prompt for permission to continue:

```
------> Continue?
```

If you respond negatively, "diskrepair" aborts. If you respond positively, it tries to continue although the amount of success it has depends on the exact nature of the damage. If the only part of the fdn which is damaged is the part that describes the type of file associated with the fdn, this repair should solve the problem. If, however, much more of the fdn is damaged, other problems will arise. In a case where the root directory is badly damaged, "diskrepair" cannot fix the disk. Nor can you salvage any information from it. Your only choice is to reformat the disk.

7.8.5 Multiple Passes

Once "fdncheck" has successfully accessed the root directory, it proceeds with checking the logical structure of the disk. Phases 2-4 of "diskrepair" are performed by "fdncheck". "Fdncheck" cannot process all the information in one pass if the number of fdns on the disk is greater than 21,240. If "fdncheck" has to make multiple passes, it prints the

following addition to the message telling you what phase it is in:

***(Pass <pass_num>)

7.9 Phase 2--Scan Directories

During phase 2 "diskrepair" scans each directory for problems in structure. A directory consists of a series of entries each containing the name of a file and the number of the fdn that describes that file. Each entry uses sixteen bytes--two for the fdn number and fourteen for the file name.

7.9.1 Size of Directory

Currently "diskrepair" cannot handle a directory of more that 4,416 entries. However, this theoretical limitation on the size of a directory is unlikely to present a practical limitation. If you do have a directory that appears to be too large, "diskrepair" prints the following message:

```
Directory too large:  <dir_name>
------> Truncate and continue?
```

If you receive this message and you know that you do not have a directory with more than 4,416 entries, chances are that the part of the fdn for that directory which contains the size is damaged. In such a case, it is safe to truncate because "diskrepair" simply sets the size to 4,416. If, on the other hand, you do have a directory that really is too large, you cannot successfully run "diskrepair" without losing the files beyond the 4,416th entry in the directory.

Because each entry in a directory contains sixteen bytes, "diskrepair" knows that the size which is stored in the fdn for a directory should be

a multiple of 16. If it is not, it reports

```
Odd size for directory "<dir_name>".
   fdn=<fdn_num>     type=d          size=<bytes>
   owner=<owner_name>  time=<time_and_date>
------> Truncate?
```

"Diskrepair" corrects this error by truncating the size stored in the fdn to the nearest multiple of 16. This repair should cause no harm.

7.9.2 Nesting Directories

"Diskrepair" can only function to the level of twelve subdirectories. If your directory structure is more deeply nested than that, "diskrepair" informs you

```
Nesting too deep at <dir_name>.
```

The program then aborts.

7.9.3 Invalid Name

While reading a directory, "diskrepair" checks the validity of the file name that appears in each entry. If the file name is invalid, it reports:

```
WARNING:  Invalid filename "<file_name>".
```

A file name is invalid if the directory entry which contains it (1) contains any non-null characters beyond the first null character in the entry or (2) contains any slash characters, '/', within the name. Although this situation is extremely unlikely to occur, the consequences are serious. The operating system simply cannot access a file with an invalid name. Therefore, it cannot be recovered. Nor can it be deleted. The file remains on the disk, inaccessible, until you reformat the disk.

7.9.4 Out-of-Range Blocks in Files

During phase 1 "diskrepair" makes a record of all the fdns that contain block addresses that are out of range. As it reads the directories during phase 2, it watches for each of these fdns. When it encounters one, it prints the following message:

```
Out-of-range block in <file_name>
   fdn=<fdn_num>      type=<char>      size=<bytes>
   owner=<owner_name>  time=<time_and_date>
------> Delete?
```

If you do not delete a file containing an out-of-range block, the structure of the disk remains damaged. However, you may want to defer deleting the file, trying first to salvage what you can (see Section 8.3.2). If neither the 'n' nor the 'p' option is in effect, "diskrepair" automatically deletes all files that contain one or more out-of-range blocks unless the file is the root directory or the file ".badblocks".

If an out-of-range block occurs in either the file ".badblocks" or in the root directory, "diskrepair" cannot delete the offending file. Rather, the program prints one of the following messages:

```
Can't delete ".badblocks".
```

or

```
Can't delete root directory.
```

To correct either of these situations you must eventually reformat the disk. Of course, if the disk is not backed up, you should try to salvage as much information as possible (see Section 8.3.2) before reformatting it.

Now consider the case where more than one directory entry points to an fdn which contains an out-of-range block (multiple links to a file). The first time "diskrepair" encounters the out-of-range block in an fdn, it asks if you want to delete the corresponding file. If you do delete the file, "diskrepair" changes the fdn number in the directory entry to 0 and changes the fdn itself so that it is inactive. Thus, the fdn numbers in the directory entries for the other files which were linked to the file you deleted now point to an inactive fdn. "Diskrepair" handles this problem as it encounters each file (see Section 7.9.8).

Suppose, however, that you deny permission when "diskrepair" asks to delete the first file corresponding to the fdn that contains an out-of-range block. In that case, "diskrepair" finds the same out-of-range block when it encounters the next file linked to that fdn. Once again, it asks for permission to delete the file. If you deny permission, the pattern repeats itself. If, on the other hand, you grant permission to delete one of the linked files after having denied it to one or more of them, you create a new problem. "Diskrepair" changes the fdn number in the directory entry to 0 and makes the fdn itself inactive. If it encounters any more files which had been linked to the deleted file, it tells you, as it should, that their directory entries point to an inactive fdn. It can fix this problem. However, the directory entries for the files you did not delete that also contain the out-of-range block now also point to the same inactive fdn. It is too late for "diskrepair" to fix this problem because it has already checked those files. It can, nevertheless, alert you to the situation and does so by printing the following message:

```
WARNING: Previous links to fdn <fdn_num>.
```

In addition, when "diskrepair" finishes, you receive another message telling you to run "diskrepair" again.

7.9.5 Blocks Duplicated in Files

During phase 1 "diskrepair" makes a record of all the fdns that contain duplicate block addresses. As it reads the directories during phase 2, it watches for each of these fdns. When it encounters one, it prints the following message:

```
Duplicate block in <file_name>.
   fdn=<fdn_num>     type=<char>     size=<bytes>
   owner=<owner_name>  time=<time_and_date>
    Duplicated in fdn <fdn_num>.
------> Delete?
```

If you do not delete all but one file containing a duplicate block, the structure of the disk remains damaged. However, at this point you do not know what other files claim the block. If you do not delete the file right away, you will find out what other files claim the same block. You can then decide whether or not you can afford to delete the necessary files. If they are not backed up, you can deny permission to delete them and try to salvage the information in them before rerunning "diskrepair" (see Section 8.3.3). Ultimately, you must delete all but

one file containing a duplicate block.

If neither the 'n' option nor the 'p' option is in effect, "diskrepair" inactivates every fdn except the last one which claims a duplicated block, unless the corresponding file is either the root directory or the file ".badblocks". The overall effect is to delete the files described by these fdns and to return to the free list all blocks they contain which are not duplicated in other files. If, however, you first execute the program with the 'n' option, "diskrepair" informs you which files contain duplicate blocks but deletes no files. If you run it with the 'p' option, you can respond 'n' to all prompts for deleting files. Once you know which files contain duplicated blocks, you can execute "diskrepair" again with the 'p' option, knowing which files you want to delete. In this way you can save the file you want to save even if it is not the last one to claim the block. In addition, you may be able to salvage some material before deleting it (see Section 8.3.3).

If a duplicate block occurs in either the file ".badblocks" or in the root directory, "diskrepair" cannot delete the offending file. Rather, the program prints one of the following messages:

```
    Can't delete ".badblocks".
or
    Can't delete root directory.
```

Since you can keep one of the files that contains a duplicate block, this situation may resolve itself cleanly--if the duplicate block really belongs in the file in question. If it does not, you will encounter other problems as well.

7.9.6 The Files "." and ".."

The first two entries in every UniFLEX directory should be the files "." and "..". The name ".." refers to the parent directory of the directory in question; the name "." refers to the directory itself. If "diskrepair" finds a directory that does not contain exactly one of each of these files, it issues one of the following messages:

```
WARNING: Too many "." entries for <dir_name>
WARNING: Too many ".." entries for <dir_name>
WARNING: No "." entry for <dir_name>
WARNING: No ".." entry for <dir_name>
```

These messages are only warnings. "Diskrepair" makes no effort to correct any of these problems. You can usually fix the case of a missing "." or ".." directory after "diskrepair" is finished (see Section 8.4). However, the case of multiple "." or ".." files cannot be repaired without deleting the directory. After deleting it, you should rerun "diskrepair", which will locate an unreferenced directory corresponding to the extra "." or ".." file. Do not place this directory in the lost-and-found directory. Instead, delete it.

If the entries "." and ".." are present in the directory, "diskrepair" checks their fdn numbers. They should match the fdns of the directory itself and its parent directory. If they do not, "diskrepair" prints whichever of the following messages is appropriate:

```
Bad "." in <dir_name>
Bad ".." in <dir_name>
```

followed by the prompt

```
-----> Fix entry?
```

It is perfectly safe to fix the entry; the change has no other effect on the disk.

7.9.7 Unknown File Type

The fdn contains information describing the type of file it refers to. Four valid types of file exist: regular file (f), directory (d), block special (b), and character special (c). If "diskrepair" does not recognize the file as one of these four types, it notifies you:

```
Unknown type for "<file_name>".
   fdn=<fdn_num>     type=u          size=<bytes>
   owner=<owner_name>  time=<time_and_date>
-----> Delete?
```

If you see this message, you may as well delete the file because it cannot be salvaged.

7.9.8 Inactive Fdn

An inactive fdn is one that does not refer to any file; it is therefore a "free" fdn. If a directory entry points to an inactive fdn, "diskrepair" reports

```
Inactive fdn for <file_name>.
   fdn=<fdn_num>     type=<char>     size=<bytes>
   owner=<owner_name>  time=<time_and_date>
------> Delete directory entry?
```

This message indicates that (1) the fdn describing the file has been severely damaged, (2) the directory entry points to the wrong fdn--which happens to be inactive, or (3) the fdn was previously cleared to fix another problem, such as an out-of-range block. In any case, you may as well delete the directory entry because "diskrepair" cannot determine what the fdn should be. If the fdn is damaged, you cannot recover the file; however, if the entry merely points to the wrong fdn, the correct fdn is almost certainly unreferenced and will, therefore, show up later in the course of repairing the disk (see Sections 7.10 and 7.11.1).

7.9.9 Out-of-range Fdns

The "format" command reserves a certain number of blocks for fdns. The fdns are numbered from 1 to the appropriate number, which varies depending on the exact form of the "format" command used. If the fdn number in any directory entry is outside this range of numbers, "diskrepair" reports

```
Out-of-range fdn for <file_name>
```

and prompts for permission to delete the directory entry. You may as well give permission because the fdn that should have been pointed to is now probably an unreferenced fdn and will show up at some other time in the process of repairing the disk (see Sections 7.10 and 7.11.1).

7.10 Phase 3--Check Unreferenced Directories

During phase 3 "diskrepair" looks for unreferenced directories. An unreferenced directory is a directory whose fdn is not pointed to by an entry in any directory in the file system. If "diskrepair" finds an unreferenced directory it sends you the message

```
Unreferenced directory.
   fdn=<fdn_num>      type=d      size=<bytes>
   owner=<owner_name>  time=<time_and_date>
Put in "lost+found"?
```

If you respond to the prompt positively, "diskrepair" names the directory "file<fdn_num>" and adds it to the directory "lost+found", informing you

```
"<file_name>" put in "lost+found".
```

and

```
Parent fdn was <fdn_num>.
```

After "diskrepair" is finished, you can look at all the files in the lost-and-found directory, decide where they belong, and reconstruct your file system.

The directory "lost+found" is normally created by the command "crdisk" when it creates a system disk. Of course, if for any reason this directory is not on the damaged disk or the directory is full, "diskrepair" cannot put any files in it. Instead of asking for permission to put the file in the lost-and-found directory, it tells you

```
Can't connect; "lost+found" directory missing or full.
------>Delete?
```

Once "diskrepair" discovers that the directory "lost+found" is full, it no longer asks you if it should put an unreferenced directory in "lost+found". Rather, it simply asks for permission to delete the file.

If your lost-and-found directory does fill up, you should let "diskrepair" finish, but do not allow it to delete any unreferenced files. Try to mount the disk and see what files are in "lost+found". If the files are not backed up, you can try to copy them to a backup device. Once you have successfully copied them, you can delete all entries in "/lost+found" and run "diskrepair" again. Once the disk is

fixed, you may want to increase the size of the lost-and-found directory (see Section 8.5).

If your disk does not have a lost-and-found directory, you cannot reliably salvage the information in the unreferenced directory or in any of the files that are descendants of that directory. If you do not have backup copies of the unreferenced files, you may try, as a last resort, to create a "lost+found" directory on the damaged disk. If you are successful, you will be able to salvage the unreferenced files. However, creating a file on a damaged disk may simply make the situation worse. You should not try to do so until you have salvaged as much information as possible from the disk and are willing to reformat it if your attempt to create the directory fails.

7.11 Phase 4--Check File and Directory Links

During phase 2 "diskrepair" builds a table which contains an entry for each fdn. The entry shows whether or not the fdn is active and, if it is, what type of file it describes. It also contains space for the link count (the number of directory entries that point to that fdn). During phase 4 "diskrepair" checks this table for certain inconsistencies.

7.11.1 Unreferenced Files

Once this table is made, it shows whether or not any unreferenced files exist. If the table shows that the link count for a file is 0, that file is unreferenced--that is, no directory entry points to that fdn. Any unreferenced files and any unreferenced directories which were not put in the lost-and-found directory in phase 3 are reported to the user as follows:

```
Unreferenced <file_or_directory>.
   fdn=<fdn_num>      type=<char>      size=<bytes>
   owner=<owner_name>  time=<time_and_date>
```

If the file is a directory, "diskrepair" asks if you want to delete it (you had a chance to put it in the lost-and-found directory in phase 3). If it is a regular file and the disk contains a lost-and-found directory that is not yet full, "diskrepair" asks

```
------>Put in lost+found?
```

If you respond negatively, "diskrepair" asks if it should delete the file. If you neither delete it nor place it in the directory "lost+found", the structure of the disk remains damaged. If you respond positively to the prompt to insert the file in the lost-and-found directory, "diskrepair" names the file "file<fdn_num>" and adds it to the directory. After "diskrepair" is finished, you can look at all the files in the lost-and-found directory, decide where they belong, and reconstruct your file system.

The directory "lost+found" is normally created by the command "crdisk" when it creates a system disk. Of course, if for any reason this directory is not on the damaged disk or the directory is full, "diskrepair" cannot put any files in it. Instead of asking for permission to put the file in the lost-and-found directory, it tells you

```
Can't connect; "lost+found" directory missing or full.
------>Delete?
```

Once "diskrepair" discovers that the directory "lost+found" is full, it no longer asks you if it should put an unreferenced file in "lost+found". Rather, it simply asks for permission to delete the file.

If your lost-and-found directory does fill up, you should let "diskrepair" finish, but do not allow it to delete any unreferenced files. Try to mount the disk and see what files are in "lost+found". If the files are not backed up, you can try to copy them to a backup device. Once you have successfully copied them, you can delete all entries in "/lost+found" and run "diskrepair" again. Once the disk is fixed, you may want to increase the size of the lost-and-found directory (see Section 8.5).

If your disk does not have a lost-and-found directory, you cannot reliably salvage the information in an unreferenced file. If you do not have backup copies of the unreferenced files, you may try, as a last resort, to create a "lost+found" directory on the damaged disk. If you are successful, you will be able to salvage the unreferenced files. However, creating a file on a damaged disk may simply make the situation worse. You should not try to do so until you have salvaged as much information as possible from the disk and you are willing to reformat it if your attempt to create the directory fails.

7.11.2 Link Count

While it is building the table, "diskrepair" keeps track of the link count for each file. If the actual link count as tallied by the process of making the table does not agree with the link count appearing in the fdn, "diskrepair" offers to fix the discrepancy:

```
Link count is <num_1>, should be <num_2>.
-----> Fix?
```

In response to a 'y', it changes the link count in the fdn to match the one determined by the tally. This change has no effect on the rest of the disk.

7.11.3 In-core Fdn List

During phase 4 "diskrepair" also examines the in-core fdn list, which is a partial list of fdns that are available for use (inactive). It tries to verify the number of free fdns in the list. If this number is in error, "diskrepair" responds

```
Bad in-core fdn count.
```

At this time "diskrepair" also checks to see if any in-core fdn which is supposedly free is actually in use. In such a case "diskrepair" tells you

```
Free in-core fdn in use.
```

Finally, "diskrepair" checks for in-core fdns that are duplicated in the list or whose fdn numbers are outside the range of permissible numbers. These errors are reported by the following messages:

```
Duplicate in-core fdns.
```

and

```
Out-of-range in-core fdn.
```

All four of these messages are followed by the prompt

-----> Fix in-core fdn list?

There is no reason to deny permission to fix the in-core fdn list because "diskrepair" can do so without affecting the rest of the disk.

7.12 Phase 5--Check Free List

During phase 5 the "diskrepair" command makes several checks on the structure of the free list. You can fix any problems encountered in this phase by rebuilding the free list (see Section 7.12.6).

7.12.1 Missing Blocks

During phase 5 "diskrepair" uses the existing free list to complete the table that it started in phase 1. This table consists of a bit map of the volume space on the disk. Each block in the volume space is represented by a particular bit. When "diskrepair" encounters a claim to a block, it sets the corresponding bit to 1. When the map is complete, all bits should be set to 1. Any bit that is still set to 0 represents a "missing" block--that is, it is neither allocated to a file nor in the free list.

7.12.2 Duplicate Blocks

If, while completing the bit map, "diskrepair" finds a bit that is already set to 1, it knows that the block represented by that bit is claimed by both the free list and a file (see Sections 7.7.3 and 7.9.5). In such a situation it prints the following informative message:

Block <block_num> duplicated in free list.

7.12.3 Out-of-Range Pointers

Occasionally, "diskrepair" reports

```
Out-of-range pointer in free list.
```

The pointer referred to has to do with the way the operating system maintains the free list. If "diskrepair" finds an out-of-range pointer, it stops checking the free list and immediately prints a summary of the information in the bit map (see Section 7.12.4). Because the summary is printed before the table is complete, the number of missing blocks is apt to be large.

7.12.4 In-core Block List

While it is checking the free list, "diskrepair" determines whether or not the number in the SIR representing the number of free in-core blocks is correct. If it is incorrect, "diskrepair" issues the following message:

```
Bad in-core block count.
```

If the in-core block count is incorrect, "diskrepair" stops checking the free list and immediately prints a summary of the information in the bit map (see Section 7.12.4). Because the summary is printed before the table is complete, the number of missing blocks is apt to be large.

7.12.5 Summary of the Status of the Free List

If "diskrepair" finds any problems during phase 5, it summarizes the status of the free list with the following messages:

```
Missing blocks = <num_of_missing_blocks>
Duplicate blocks in free list = <num_of_dupl_blocks>
Out-of-range blocks in free list = <num_of_out_of_range_blocks>
```

7.12.6 Rebuilding the Free List

After reporting any errors found in phase 5, "diskrepair" prompts for permission to rebuild the free list:

```
Free list bad.
------> Rebuild?
```

"Diskrepair" reconstructs the free list from the information in the bit map. Rebuilding the free list does not affect any data on the disk. Thus, it is a safe procedure unless "diskrepair" was unable to read the file ".badblocks" (see Section 7.6). On a large disk, however, the process may take a considerable amount of time (as much as fifteen minutes). If the only problem is missing blocks, which are harmless, you may wish to wait and rebuild the free list at a time that will not inconvenience too many people.

If you give permission for "diskrepair" to rebuild the free list, it enters phase 5B. During this phase it places in the free list every block that is in the volume space but is not allocated to a file and is not in the file ".badblocks". When it finishes rebuilding the free list, it prints the message

```
Free list rebuilt (<num> blocks)
```

where <num> is the number of blocks in the reconstructed free list.

7.13 Phase Six--Check SIR Information

During its final phase of operation, "diskrepair" makes several more checks on the System Information Record.

7.13.1 Free Fdn Count

If the number stored in the SIR which represents the total number of free fdns on the disk does not agree with the number calculated by "diskrepair" as it moves through the fdns one by one, "diskrepair" reports

```
Free fdn count in SIR wrong.
------>Fix?
```

In response to a positive answer, "diskrepair" changes the number in the SIR. This change has no effect on anything else on the disk.

7.13.2 Free Block Count

Similarly, if the number stored in the SIR which represents the total number of free blocks on the disk does not agree with the number calculated by "diskrepair" as it checks allocated blocks, "diskrepair" reports

```
Free block count in SIR wrong.
------>Fix?
```

In response to a positive answer, "diskrepair" changes the number in the SIR. This change has no affect on anything else on the disk.

7.13.3 State of the Disk

If "diskrepair" modifies the disk, it prints the following message on completion of its tests:

```
=== Disk modified. ===
```

If it does not modify the disk and the 'v' option was not in effect, it prints whichever of the following messages is appropriate:

```
=== Disk OK. ===

=== Disk needs repair! ===
```

If you did specify the 'v' option, "diskrepair" assumes that its output has already indicated whether or not you need to repair the disk.

"Diskrepair" may encounter more problems than it can fix during one run. For example, it can only handle a certain number of duplicate or out-of-range blocks. If "diskrepair" cannot fix all the errors it encounters, or if it encounters a read or write error but you choose to continue operation, it prints the following message when it is finished:

=== Problems encountered. Diskrepair should be rerun. ===

If appropriate, this message occurs no matter what options you specify.

If the 'a' option is in effect and the disk is so badly damaged that all the iterations of "diskrepair" fail to fix it, the following message appears;

=== Repair of "<dev_name>" not complete after 10 tries. ===

7.13.4 Updating the SIR

When it is finished modifying the disk, "diskrepair" must update the SIR so that it corresponds to the new structure of the disk. If the device being checked is not the root device, "diskrepair" simply rewrites the SIR with the correct information. If, however, the SIR of the root device must be updated, "diskrepair" kills all tasks running on the system and locks up the system so that no new tasks can begin. It then modifies the SIR. This procedure is necessary to prevent conflicts between the written data and similar data kept in memory. After updating the SIR, "diskrepair" stops the system and prints the following message:

=== Intentional system stop. Reboot UniFLEX. ===

If you receive this message, you must reboot the system before you can proceed.

If "diskrepair" encounters an I/O error when it tries to make any changes in the SIR, it prints the following message:

ERROR UPDATING SIR. DISK IS BAD!

This error is not only fatal to "diskrepair"; it also means that you

cannot salvage any of the data on your disk.

7.14 I/O Errors

If "diskrepair" encounters an I/O error, it first prints a generic message telling you that it could not read from or write to the disk. It then prompts you for permission to continue and, if you respond positively, usually gives you a more detailed message which tells you what it was trying to do when it encountered the error (see Section 7.1.2). We suggest that you respond negatively to the prompt to continue the first time "diskrepair" reports an I/O error and that you immediately rerun "diskrepair". It is possible--though unlikely--that the I/O error is a soft one and will not recur. If you receive the same error message again, rerun "diskrepair" with the 'a' option.

An alphabetic list of the I/O error messages follows. An explanation accompanies each message.

Can't fix duplicated block.
"Diskrepair" found a duplicated block in an fdn, but when it tried to read the fdn to find out the owner, the size, and so forth, it encountered an I/O error.

Can't fix inactive fdn.
"Diskrepair" found a directory entry which pointed to an inactive fdn, but when it tried to read the fdn to find out the owner of the file, the size, and so forth, it encountered an I/O error.

Can't fix link count error.
"Diskrepair" encountered an I/O error while trying to read the link count from an fdn.

Can't fix out-of-range block.
"Diskrepair" found an out-of-range block in a file, but was unable to read the fdn to find out the owner, size, and so forth.

Can't fix unknown file.
"Diskrepair" encountered an I/O error trying the read the fdn of a file whose type was unknown.

Can't fix unreferenced fdn.
"Diskrepair" found an unreferenced file or directory, but got an I/O error while trying to read the fdn to determine the owner, size, and so forth.

Can't truncate directory.
"Diskrepair" tried to truncate the size of a directory, but it encountered an I/O error when it tried to write the new size in the fdn.

Connect to "lost+found" unsuccessful.
"Diskrepair" was unable to write to the directory ".lost+found" and therefore could not put the unreferenced fdn into that directory. If you receive this error message, "diskrepair" cannot put any files in "/lost+found". Eventually, you must delete the unreferenced files, but you may be able to salvage some data first (see Chapter 8).

Error opening <dir_name>.
Directory completely ignored!
"Diskrepair" was trying to scan all the directories on the disk, but due to an I/O error it was unable to open the directory <dir_name>. All files and subdirectories in that directory, as well as any files in the subdirectories (and so forth) are now unreferenced.

Directory partially ignored.
"Diskrepair" was trying to scan all directories on the disk, but due to an I/O error in one of the blocks in <dir_name> it was unable to read the entire directory. It therefore lacks information on the directory entries that were maintained in that particular block and in all subsequent blocks in <dir_name>. Any descendents of these files are now unreferenced.

Fdn not updated.
"Diskrepair" encounted an I/O error when it tried to write to an fdn.

Fdns <fdn_num_1> to <fdn_num_2> skipped.
"Diskrepair" encountered an I/O error while reading a block of fdns. It is, therefore, unable to read any of the fdns in that block. Since each block contains eight fdns, as many as eight files may be inaccessible. Such a situation usually causes other problems, but you may be able to salvage some data. If the error is in the first fdn block, you must reformat the disk. You cannot salvage any data from the disk.

File or directory not deleted.
"Diskrepair" tried to delete a file or a directory but was unable to write to the corresponding fdn.

Free list check aborted.
"Diskrepair" encountered an I/O error while trying to read the free list. It stops trying to read the free list, prints the message, "Free list bad.", and tries to rebuild the free list. The free list may, however, contain a bad block. When "diskrepair" is done, you should run "devcheck" to determine which block or blocks are bad and use "badblocks" to put those blocks in the bad-blocks file.

Links for connected file may be bad.
When "diskrepair" puts a file in the lost-and-found directory, it tries to correct the link count. This message appears if "diskrepair" encountered an I/O error while trying to fix the link count.

Omitting block <block_num> from free list.
In the process of rebuilding the free list, "diskrepair" encountered an I/O error. It leaves the offending block out of the free list; therefore, that block becomes a missing block. This situation causes no immediate problems; however the next time you run "diskrepair" the bad block may be put back in the free list, thus creating the potential for an I/O error during normal operation of the operating system. You should use the "badblocks" command to put the offending block in the file "/.badblocks".

Part of file may be ignored.
The operating system encountered an I/O error while trying to read an fdn during phase 1. As a result "diskrepair" may not be aware of all the blocks that are supposed to be in the file and may release them to the free list.

WARNING: Link count and reference errors may be invalid!
This error results only from a highly unlikely arrangement of files. If you receive this error, contact Technical Systems Consultants for assistance.

7.15 Index of Error Messages

This section contains an index to the error messages which are not caused by I/O errors.

Nesting too deep at <dir_name>, 7.22
No device specified, 7.9
No such device, 7.9
Not a block device, 7.9

Odd size for directory "<dir_name>", 7.22
Out-of-range block in fdn <fdn_num>, 7.17
Out-of-range block in <file_name>, 7.23
Out-of-range blocks in free list = <num_of_out_of_range_blocks>, 7.33
Out-of-range fdn for <file_name>, 7.27
Out-of-range in-core fdn, 7.31
Out-of-range pointer in free list, 7.33
Output directed to device under test, 7.12

Permission denied, 7.10
Problems encountered. Diskrepair should be rerun, 7.36

Repair of "<dev_name>" not complete after 10 tries, 7.36
Root directory zero length, 7.20
Root fdn is not a directory, 7.20

SIR fdn block count error: <num_of_fdns>, 7.14

Too many duplicate blocks, 7.18
Too many out-of-range blocks, 7.17

Unknown option: <char>, 7.9
Unknown type for "<file_name>", 7.26
Unmount error: <error_num>, 7.11
Unreferenced directory, 7.28
Unreferenced <file_or_directory>, 7.29

WARNING: Invalid filename in "<dir_name>", 7.22
WARNING: No "." entry for <dir_name>, 7.25
WARNING: No ".." entry for <dir_name>, 7.25
WARNING: Possible file size error in fdn <fdn_num>, 7.16
WARNING: Previous links to fdn <fdn_num>, 7.24
WARNING: Too many ".." entries for <dir_name>, 7.25
WARNING: Too many "." entries for <dir_name>, 7.25

Chapter 8

Recovering from a Damaged Disk

8.1 Introduction

Your best insurance against the loss of data due to a damaged disk is a good backup system (see Section 1.12). The utilities "copy-dir" (in Utilities Package I) and "backup" (in Utilities Package II) can help you to streamline your backup procedures so that you need not use lots of valuable time backing up your system.

8.2 Restoring Files from the UniFLEX Master Disk

Sometimes when a system crashes or a program malfunctions, some files or directories are destroyed. You can always recreate the original UniFLEX system by executing the "crdisk" command. However, if the damage is slight, you may choose to recreate only those files or directories which were damaged. To do so, simply mount the master disk for read only in floppy disk drive 1 using the following command:

```
/etc/mount /dev/fd1 /usr2 r
```

Then copy the necessary files to your system disk.

8.2.1 Who Owns What?

One problem with this approach is that the "copy" command always makes the user copying the file the owner of the new copy. Thus, all the files you copy when you are logged in as system manager are owned by "system". In order for the system to function properly, certain files must be owned by "system"; others, by "bin". A breakdown of the ownership of files by directory, as supplied on the master disk, follows:

1. /act
   The directory "/act" and all the files in it are owned by "system".

2. /bin
   The directory "/bin" is owned by "bin", as are all the files in it except "crdir", "mail", and "path", which are owned by "system".

3. /dev
   The directory "/dev" and all the devices (files) in it, with the exception of the terminals, are owned by "system". A terminal is owned by the user who is currently logged in on it. If the terminal is not in use, it is owned by system.

4. /etc
   The directory "/etc" is owned by "system", as are all the files in it except "prcon", which is owned by "bin". All the files in the directory "/etc/log" are also owned by "system".

5. /gen
   The directory "/gen" is owned by "bin". The directory "/gen/help" and the files it contains are also owned by "bin". The directory "/gen/errors" and the file it contains are owned by "system".

6. /lib
   The directory "/lib" and all the files it contains are owned by "bin".

7. /lost+found
   The directory "lost+found" is owned by "system". Any files that "diskrepair" places there belong to the original owner.

8. /tmp
   The directory "/tmp" is owned by "bin". Any files placed in "/tmp" belong to the user who creates them.

9. /usr
   The directory "/usr" is owned by "bin", as is the directory "/usr/bin". All files in "/usr/bin" are owned by "bin" except the file "password", which is owned by "system". The directory "/usr/spooler" is owned by "system".

10. /usr2
    The directory "/usr2" is owned by "bin".

8.2.2 Setting s+ Permissions

You are already familiar with the read, write, and execute permissions which are shown when you execute the command "dir +1" (see 6809 UniFLEX: A Tutorial). You can also set another type of permission, called 's' permission, with the "perms" command. If the 's' permission bit is on for a given program, any user executing that program has the same privileges as the owner of the program for the duration of the task.

Certain UniFLEX commands do not function if the 's' permission bit is not set. Therefore, if you recreate any files, be sure that you use the "perms" command to set the 's' bit on the following files:

1. /etc/login
2. /etc/print
3. /usr/bin/password
4. /usr/bin/crdir
5. /usr/bin/mail
6. /usr/bin/path

The correct form of the command is

perms s+ <file_name> [<file_name_list>]

8.3 Salvaging Data from a Damaged Disk

No matter how thorough you are about backing up your disk, it is possible that at some point you will experience a system crash when you are not completely backed up. When you run "diskrepair", you may discover that the only way to repair the disk is to destroy files that are not backed up. This section is designed to help you recover as much data as possible in such a circumstance.

8.3.1 General Salvage Procedure

If for any reason "diskrepair" cannot fix a problem, you will eventually have to reformat your disk. Before you do so, you may be able to salvage some of the data by using the following procedure.

1. Use a disk that is intact to boot the system.

2. Mount the damaged disk for reading only with the following command:

   /etc/mount <dev_name> <dir_name> r

3. Copy as much information as possible from the damaged disk to a backup device.

The extent to which this procedure is successful depends on the nature of the damage to the disk.

8.3.2 Out-of-range Block in a File

If a file contains an out-of-range block, you cannot access the entire file. At the very least, that part of the file that is associated with the out-of-range block is inaccessible. If the file is a binary file, you cannot salvage any of it. If, however, the file is a text file, the rest of it can be recovered using the "head" and "tail" utilities (in Utility Package I).

First of all, boot the system with an intact disk and mount the damaged disk for reading only with the following command:

/etc/mount <dev_name> <dir_name> r

To salvage the file, use the "head" command repeatedly with different arguments until you find the largest argument that you can successfully use. Then, execute the command with this argument and redirect the output to a temporary file on the undamaged disk.

Repeat the procedure for the "tail" command. Note that, because a file always starts at the beginning of a block, the "head" command fails with an argument whose value is a multiple of 512 (the number of bytes in a block) plus 1. You cannot predict, however, where the "tail" command will fail because the end of the file is not necessarily the end of a block. Once you have found the limit of the "tail" command, repeat that command and append the output to the temporary file containing the first part of the file.

The temporary file now contains all but one block of the original file. You cannot salvage the data associated with the out-of-range block, but you have recovered the rest of the file. Now you can rerun "diskrepair" and allow it to delete the file containing the out-of-range block. When "diskrepair" is complete, rename your temporary file with the name of the original file and move it back to the appropriate directory on the newly repaired disk. The file is not complete, but it is a lot better than no file.

8.3.3 Duplicate Blocks in a File

You should be able to back up any files that contain duplicate blocks by following the procedure given in Section 8.3.1 unless the disk is too badly damaged to be mounted. One of the files is probably still intact. The others are missing a block of their original data, which has been replaced by the duplicate block.

8.4 Fixing Missing "." and ".." Files

In general, the situation in which a "." or a ".." file is missing is simple to fix. To create the correct "." file you simply link the directory in question to the file "." in that directory. Similarly, to create the correct ".." file you link the parent of the directory in question to the ".." file in the directory.

For example, if the directory "/usr/larry/tests" is missing both the "." and ".." entries, you can probably correct the situation with the following two commands:

```
link /usr/larry/tests /usr/larry/tests/.
link /usr/larry /usr/larry/tests/..
```

8.5 Expanding the Directory "lost+found"

When the operating system creates a directory, it allocates one block for it. Since each directory entry uses 16 bytes, this first block can hold thirty-two directory entries. However, in the case of the first

block of a directory, the first two entries are used for the files "." and "..". Thus, a newly created directory has room for thirty entries before another block must be allocated to that directory.

The directory "lost+found" is created by the program "crdisk". When the directory is first created, it--like any other newly created directory--has room for thirty entries. If you wish to increase the number of entries it can hold, use the following procedure. You should not try to increase the size of the lost-and-found directory on a damaged disk.

1. Boot the system with the disk containing the lost-and-found directory whose size you want to increase.

2. Change directories using the command

   chd /lost+found

3. Create as many files as necessary to force the operating system to allocate another block to the directory. If the directory is empty (except for the "." and ".." entries) and is one block long, you must add thirty-one files to allocate another block to the directory. After that you must add thirty-two files for each block you want to add.

4. Verify that the directory is as large as you want by issuing the command

   dir +1 /

   Included in the information shown by this command is the size of the directory in blocks.

5. Delete all the files you just created. The number of unreferenced files that "diskrepair" can fit into the directory is

   (<size_in_blocks> * 32) - 2

Appendix A

Syntax Conventions

The following conventions are used in syntax statements throughout this manual.

1. Items that are not enclosed in angle brackets, '<' and '>', or square brackets, '[' and ']', are "keywords" and should be typed as shown.

2. Angle brackets, '<' and '>', enclose descriptions which the user must replace with a specific argument. For example, in the line

   ```
   spr <file_name>
   ```

   the name of a file must be specified in the place indicated by <file_name>.

3. Square brackets, '[' and ']', indicate optional items. These items may be omitted if their effect is not desired.

4. The underscore character, '_', is used to link separate words that describe one term, such as "file" and "name".

5. Characters other than spaces that are not enclosed in angle brackets or square brackets must appear in the precompiler statement as they appear in the syntax statement.

6. If the word "list" appears as part of a term in a syntax statement, that term consists of one or more of the elements described by the rest of the term, separated by commas or spaces. For example, the term

   ```
   dev_name_list
   ```

   represents a list of valid device names.

7. Many UniFLEX commands support one or more optional features, known as options, which alter the effect of the command. Options consist of either a single character or a single character followed by an equals sign, '=', followed by an argument. An "option string" is a plus sign followed by one or more options. An option string may contain any

number of single-character options but only one option which takes an argument. An option requiring an argument must be the last option in an option string. Thus, the command line must contain a separate option string for each option requiring an argument. It may or may not contain a separate option string for each single-character option.

The following are valid option strings:

```
+abcdefg
+abc=<arg>
```

The following are invalid option strings:

```
+abc=<arg>de
+a=<arg>b=<arg>
```

Unless specifically stated in the documentation about a particular command, options strings may appear anywhere on the command line after the command name.

Many common terms appear (often as abbreviations) in more than one syntax statement. The manual does not explain these terms each time they appear; however, the following table describes each one.

Table A-1. Common Terms Used in Syntax Statements

Term	Meaning
char	Character
dev	Device
dir	Directory
dupl	Duplicate
fdn	File descriptor node
file_name	A valid file name
list	Term is a list of elements
num	Number
param	Parameter
str	String
splr	Spooler
user_name	A valid user name