ULTRIX-11™
System Management Guide

Order No. AA-X343B-TC

digital equipment corporation, maynard, massachusetts

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a Digital Equipment Corporation license and may be used or copied only in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by Digital Equipment Corporation or its affiliated companies.

The following are trademarks of Digital Equipment Corporation:

DEC	Edusystem	ULTRIX
DEC/CMS	IAS	ULTRIX-11
DEC/MMS	MASSBUS	ULTRIX-32
DECnet	MICRO/PDP-11	ULTRIX-32m
DECsystem-10	Micro/RSX	UNIBUS
DECsystem-20	PDP	VAX
DECUS	PDT	VMS
DECwriter	RSTS	VT
DIBOL	RSX	digital™

The following command descriptions as set forth in this document are copyrighted material of Digital Equipment Corporation:

bufstat(1m)	osload(1m)
cda(1m)	rasize(1m)
cdc(1)	rx2fmt(1m)
chog(1)	sysgen(1m)
chroot(1)	ted(1)
csf(1m)	tss(1m)
lpset(1m)	usat(1)
memstat(1m)	zaptty(1m)

In accordance with the licenses granted to Digital Equipment Corporation by AT&T Bell Laboratories and the University of California at Berkeley pertaining to the software described herein, the following should be understood by the licensee, and any related documentation provided by the licensee to third parties, whether pursuant to an agreement with Digital Equipment Corporation permitting sublicensing of the software described herein or otherwise, must contain the provisions of this section as set forth below:

a Information herein is derived from copyrighted material as permitted under a license agreement with AT&T Bell Laboratories.
Copyrighted © 1979 AT&T Bell Laboratories.

"Make – A Program for Maintaining Computer Programs" acknowledgements: S.C. Johnson, and H. Gajewska, for their ideas and assistance.

"Screen Updating and Cursor Movement Optimization: A Library Package" acknowledgements: For their help and support, Bill Joy, Doug Merritt, Kurt Shoens, Ken Abrams, Alan Char, Mark Horton, and Joe Kalash.

"YACC: Yet Another Compiler-Compiler" acknowledgements: B.W. Kernighan, P.J. Plauger, S.I. Feldman, C. Imagna, M.E. Lesk, A. Snyder, C.B. Haley, D.M. Ritchie, M.D. Harris and Al Aho, for their ideas and assistance.

"Lex – A Lexical Analyzer Generator" acknowledgements: S.C. Johnson, A.V. Aho, and Eric Schmidt, for their help as originators of much of Lex, as well as debuggers of it.

The document "RATFOR – A Preprocessor for a Rational Fortran" is a revised and expanded version of the one published in Software – Practice and Experience, October 1975. The Ratfor described here is the one in use on UNIX and GCOS at AT&T Bell Laboratories. Acknowledgements: Dennis Ritchie, and Stuart Feldman, for their ideas and assistance.

"The M4 Macro Processor" acknowledgements: Rick Becker, John Chambers, Doug McIlroy, and Jim Weythman, for their help and support.

"BC – An Arbitrary Precision Desk-Calculator Language" acknowledgement: The compiler is written in YACC; its original version was written by S.C. Johnson.

"A Dial-Up Network of UNIX TM Systems" acknowledgements: G.L. Chesson, A.S. Cohen, J. Lions, and P.F. Long, for their suggestions and assistance.

The document "Berkeley Pascal User's Manual" is copyrighted © 1977, 1979, 1980, 1983 by W.N. Joy, S.L. Graham, C.B. Haley, M.K. McKusick, P.B. Kessler. The financial support of the first and second authors' work by the National Science Foundation under grants MCS74-07644-A04, MCS78-07291, and MCS80-05144, and the first author's work by an IBM Graduate Fellowship are gratefully acknowledged.

d The Fourth Berkeley Software Distribution is provided by the Regents of the University of California and the Other Contributors on an "as is" basis. Neither the Regents of the University of California nor the Other Contributors warrant that the functions contained in the Fourth Berkeley Software Distribution will meet the licensee's requirements or will operate in the combinations which may be selected for use by the licensee, or that the operation of the Fourth Berkeley Software Distribution will be uninterrupted or error free. Neither the Regents of the University of California nor the Other Contributors make any warranties, either express or implied, as to any matter whatsoever, including without limitation, the condition of the Fourth Berkeley Software Distribution, its merchantability or its fitness for any particular purpose.

e The licensee understands and agrees that the Regents of the University of California and the Other Contributors are under no obligations to provide either maintenance services, update services, notices of latent defects, or corrections of defects for Fourth Berkeley Software Distribution.

b UNIX is a trademark of AT&T Bell Laboratories.
The UNIX trademark may not be used in the name of any licensee's product. Any use of the trademark in advertising, publicity, packaging, labelling or otherwise must state that UNIX is a trademark of AT&T Bell Laboratories.

c This software and documentation is based in part on the Fourth Berkeley Software Distribution under license from the Regents of the University of California. We acknowledge the following individuals and institutions for their role in its development:

The Electrical Engineering and Computer Sciences Department at the Berkeley Campus of the University of California. Ken Arnold, Earl T. Cohen, John Foderaro, Charles Haley, Mark Horton, William Joy, Jim Kleckner, Geoffrey Peck, Cliff Matthews, University of New Mexico; Eric Shienbrood.

"The UNIX Time-Sharing System": Copyright © 1974, Association for Computing Machinery, Inc. reprinted by permission. This is a revised version of an article that appeared in Communications of the ACM, 17, No. 7 (July 1974), pp.365-375. That article was a revised version of a paper presented at the Fourth ACM Symposium on Operating Systems Principles, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, October 15-17, 1973. Acknowledgements: for their help and support, R.H. Canaday, R. Morris, M.D. McIlroy, and J.F. Ossanna.

"Advanced Editing on UNIX" acknowledgement: Ted Dolotta for his ideas and assistance.

"Ex Reference Manual" acknowledgements: Chuck Haley contributed greatly to the early development of ex. Bruce Englar encouraged the redesign which led to ex version 1. Bill Joy wrote versions 1 and 2.0 through 2.7, and created the framework that users see in the present editor. Mark Horton added macros and other features and made the editor work on a large number of terminals and UNIX systems.

"An Introduction to the UNIX Shell" acknowledgements: Dennis Ritchie, John Mashey and Joe Maranzano for their help and support.

"An Introduction to the C Shell" acknowledgements: Michael Ubell, Eric Allman, Mike O'Brien, and Jim Kulp.

"LEARN – Computer-Aided Instruction on UNIX" acknowledgements: for their help and support, M.E. Bittrich, J.L. Blue, S.I. Feldman, P.A. Fox, M.J. McAlpin, E.Z. Rothkopf, Don Jackowski, and Tom Plum.

"A System for Typesetting Mathematics" acknowledgements: J.F. Ossanna, A.V. Aho, and S.C. Johnson, for their ideas and assistance.

"A TROFF Tutorial" acknowledgements: J.F. Ossanna, Jim Blinn, Ted Dolotta, Doug McIlroy, Mike Lesk and Joel Sturman, for their help and support.

The document "The C Programming Language – Reference Manual" is reprinted, with minor changes, from "The C Programming Language", by Brian W. Kernighan and Dennis M. Ritchie, Prentice-Hall, Inc., 1978.

Table of Contents

TABLES

FIGURES

Appendixes

Introduction: ULTRIX-11 Documentation

The documentation for the ULTRIX-11 operating system is divided into two sets. The first set, user-level documentation, consists of:

- ULTRIX-11 Programmer's Manual, Volume 1
- ULTRIX-11 Programmer's Manual, Volume 2A
- ULTRIX-11 Programmer's Manual, Volume 2B

The second set, system-level documentation, consists of:

- ULTRIX-11 Software Technical Description
- ULTRIX-11 Installation Guide
- ULTRIX-11 System Management Guide

ULTRIX-11 System Management Guide

This document assumes the reader has a basic understanding both of the UNIX operating system and of the differences between the UNIX Time-Sharing System, Seventh Edition and the ULTRIX-11 software. If you do not have such understandings, you should read the ULTRIX-11 Software Technical Description and the following documents from the ULTRIX-11 Programmer's Manuals:

- UNIX for Beginners - Second Edition
- The UNIX Time-Sharing System
- An Introduction to the UNIX Shell
- The UNIX I/O System
- UNIX Implementation

*UNIX is a trademark of AT&T Bell Laboratories

Chapter 1

ULTRIX-11 I/O System

To administer an ULTRIX-11 system, the system manager should have an understanding of how the ULTRIX-11 I/O system works and how the operating system communicates with devices. As system manager, you should read this chapter first and then read the remaining chapters as the need arises.

The remaining sections of this chapter discuss:

- I/O system calls
- ULTRIX-11 terminology
- Logical partitioning of disks
- Logical device names and special files
- Disk unit numbers

1.1 I/O System Calls

The ULTRIX-11 operating system uses devices for:

- Loading programs
- Swapping process images
- Manipulating files

In addition, the ULTRIX-11 operating system lets user-level programs use devices for accessing files. In performing I/O, a user-level program uses system calls:

- close() - Close files
- creat() - Create and open files for writing
- lseek() - Position file access pointer
- open() - Open existing file for reading/writing
- read() - Read data from an open file
- write() - Write data to an open file

1.2 ULTRIX-11 Terminology

The following terms relate to ULTRIX-11 I/O operations and are used throughout this manual.

1.2.1 ULTRIX-11 Kernel

The ULTRIX-11 kernel is the memory-resident portion of the system that schedules processes, services system calls, maintains the file systems, and interacts directly with the system hardware.

1.2.2 File System

To the ULTRIX-11 kernel, a file system is a 512-byte block structure that is imposed on a disk or disk partition. This disk-resident structure lets the ULTRIX-11 kernel store and access files. Essentially, a file system can be divided into four regions:

- Block 0 is the boot block. On the root file system, the boot block contains the primary boot program. On non-bootable file systems, the boot block usually is unused. For further information, read Section 3.1, The Boot Sequence.

- Block 1 is the superblock. On each file system, the superblock contains the parameters that define the physical structure of that file system. For example, the superblock defines the size of the file system as well as the starting and ending block addresses of the file system, the inode list (ilist), the free inode list, and the free block list.

- Block 2 through block n is the ilist. On each file system, the ilist is a series of 64-byte structures called inodes. Both the number of blocks allocated to and the number of inodes contained in the ilist varies by the size of the file system. Each allocated inode contains the parameters that define a directory, a special file, or a data file in that file system.

- Block n+1 through the last block in the file system are the free data blocks. On each file system, blocks are allocated as needed to directories and data files from the chained free list, a series of blocks that each contain the addresses of 100 free blocks.

Figure 1-1 illustrates the 4-region file system layout.

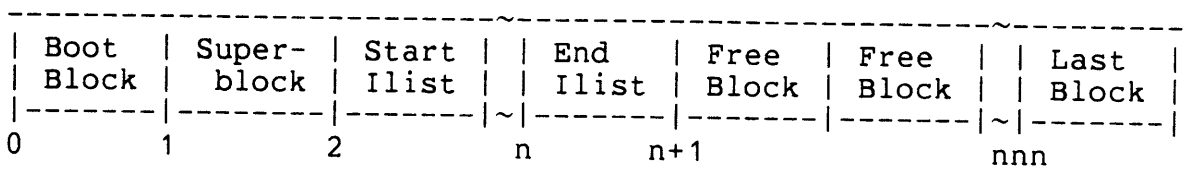


Figure 1-1 File System Layout

For further information, read filsys(5) in the ULTRIX-11 Programmer's Manual , Volume 1.

1.2.3 Single-User Mode

During single-user mode, the ULTRIX-11 system runs with only the root file system mounted and with the superuser account active (running /bin/sh) at the console. No other terminals are enabled, and no other login accounts are active.

Because the login command was not invoked, however, the normal user environment was not fully created for the superuser account. Specifically, the HOME, TERM, SHELL, and USER variables have not been set. Under certain circumstances, this can cause unexpected errors. For example, errors occur when using either the cd command without specifying a directory or the vi command at the console terminal.

1.2.4 Multiuser Mode

When multiuser mode is invoked, the ULTRIX-11 system automatically reads the multiuser start-up file, /etc/rc. This file is a shell script that lists those commands that, when executed, prepare the system for time sharing. The following provides a line-by-line explanation of the /etc/rc file:

- Sets default system-level PATH shell variable
- Removes the /etc/mtab file (mount table)
- Initializes the /etc/utmp file
- Mounts all file systems listed in /etc/fstab
- Removes the login lock file
- Enables system accounting (optional)
- Cleans up /usr/tmp
- Preserves ex and vi temporary files (if any)
- Cleans up /tmp
- Removes all tip and uucp lock files
- Invokes the update command
- Invokes the cron command
- Enables system error logging
- Displays the size of the error log file (optional)
- Enables terminals
- Restarts the line printer spooler (if interrupted)

When multiuser mode is restarted using the opser command, the ULTRIX-11 system automatically invokes the multiuser

restart file, /opr/restart. This file also is a shell script that lists those commands that, when executed, prepare the system for time-sharing. The following provides a line-by-line explanation of the /opr/restart file:

- Removes the /etc/mtab file (mount table)
- Mounts all file systems listed in /etc/fstab
- Enables system accounting (optional)
- Cleans up /usr/tmp
- Cleans up /tmp
- Removes all tip and uucp lock files
- Enables system error logging
- Displays the size of the error log file (optional)
- Invokes the cron command
- Invokes the update command
- Removes the login lock file
- Restarts the line printer spooler (if interrupted)
- Enables terminals

For further information, read Chapter 3, ULTRIX-11 Boot Procedures, Section 5.1.6, Restarting Multiuser Mode, as well as cron(8) and update(8) in the ULTRIX-11 Programmer's Manual, Volume 1.

1.2.5 Mounting File Systems

Mounting a file system creates a logical association (link) between that file system and the directory on which it is mounted. Without this logical link, the system and users could not access data from the file system.

You can mount a file system either on an empty directory or on one that contains existing entries for files. When you mount a file system on a directory that contains existing entries, however, you make the corresponding files inaccessible. While the files in the mounted file system become accessible, the previously existing files logically disappear. When you unmount the file system, the previously existing files again become accessible.

When you create files in a directory on which a file system is mounted, those files logically exist in the root directory of the mounted file system. They do not exist in the directory on which the file system is mounted.

1.2.6 File

To the average user, a file is a collection of related information. To the user-level program that manipulates this information, a file is a contiguous stream of bytes. To the ULTRIX-11 kernel, a file is a series of 512-byte blocks.

The user-level program that creates a file determines how

its data is organized. The ULTRIX-11 system simply stores this data as a contiguous string of bytes, which it accesses in 512-byte blocks. A user-level program accesses a file with an access pointer set to the byte address by using the lseek() system call. For further information, read lseek(2) in the ULTRIX-11 Programmer's Manual, Volume 1.

1.2.7 Special File

A special file is an entry in the /dev directory that the ULTRIX-11 system uses to access a configured I/O device. Each configured device has at least one associated special file. When a user-level process accesses a special file, the system activates the associated device. For example, when a user-level process writes data to /dev/lp (line printer spooler), this data is printed on the system line printer.

1.2.8 Directory

A directory is a file that contains a 16-byte entry for every file, special file, and directory assigned to it. Essentially, the directories in a file system logically organize it into groups of related files and give it its tree structure. Each directory entry contains:

- Inumber (2 bytes)
- File name (14-byte maximum)

The first two entries in every directory (. and ..) are special names. The entry for . lists the inumber of the directory itself. The entry for .. lists the inumber of the directory in which its own entry is contained (parent directory).

1.2.9 Inumber

An inumber is an inode index number. Although a user specifies a file name when accessing a file, the system automatically translates that name to the corresponding inumber and uses it to access the appropriate inode.

1.2.10 Inode

An inode is a 64-byte data structure that contains the parameters that define a file, special file, or directory. Each inode lists:

- Type (regular file, special file, or directory)
- Mode (access permissions)
- Owner (userID)
- Group (groupID)
- Size (in bytes)
- Block addresses (directory and regular file)
- Major/minor numbers (special file)
- Last time accessed
- Last time modified

For further information, read filsys(5) in the ULTRIX-11 Programmer's Manual , Volume 1.

1.2.11 File Mode Settings

Regardless of the type, each inode (file) has nine permission bits (three sets) which the ULTRIX-11 operating system uses in determining what access permissions (mode) apply for a given user. The first set determines read, write, and execute permission for the file's owner. The second set determines read, write, and execute permission for all group members. The last set determines read, write, and execute permission for all other users.

When a user attempts to access a directory, special file, or data file, the ULTRIX-11 system first opens the appropriate inode and compares the user's ID number with the userID listed in the inode. If these userIDs match, the system uses the first set of permission bits and grants "owner" access. If they do not match, the system then compares the user's groupID with the groupID listed in the inode. If these groupIDs match, the system uses the second set of bits and grants "group" access. If neither userIDs nor groupIDs match, the system defaults to the third set of bits and grants "other" access.

1.2.12 Directory Mode Settings

Although each directory inode has the standard nine permission bits, the ULTRIX-11 system uses them differently than those for special files or data files. Each read bit specifies permission to list the contents (entries) of the directory. Each write bit specifies permission to create an entry in the directory. Each execute bit specifies permission to search for or access an inumber from an entry in the directory.

1.2.13 UserID

A userID is the unique number by which the ULTRIX-11 kernel identifies a user's processes and determines owner access permission to files. Although a user specifies a login name when logging in, the system automatically translates this

name to the appropriate userID.

1.2.14 GroupID

A groupID is the identification number by which the ULTRIX-11 kernel determines group access permission to files. In addition to translating the login name to a userID during the login process, the system establishes the user's groupID.

1.2.15 Block I/O Mode

In block (buffered) I/O mode, the ULTRIX-11 kernel uses the I/O buffer cache to transfer data between a user process and a device. Normally, the ULTRIX-11 kernel uses block I/O mode whenever it accesses files.

In block I/O mode, the user-level process opens the desired file and specifies the byte offset from the start of the file and the number of bytes that are to be transferred. The system then converts the byte offset into a block number and attempts to locate that block in the buffer cache. If that block is not already in the buffer cache, the system loads it in from disk.

On a read operation, the system passes the requested number of bytes from the buffer cache to the user-level process's buffer. Anticipating that this process will read the next block, the system automatically initiates a read on the next block. This feature is called read ahead.

On a write operation, the system passes the desired number of bytes from the user-level process's buffer to the buffer cache. In anticipation of further writes to the same block, the system does not write that block out to the file system immediately. Instead, the block remains in the buffer cache until the ULTRIX-11 kernel performs a sync or flushes that buffer. This is known as delayed write or write behind.

NOTE

To prevent file system inconsistencies, the system writes out immediately all blocks that contain inode information.

During normal operations, the ULTRIX-11 system flushes the buffer cache whenever one of the following occurs:

- The update program invokes the sync system call--normally every 30 seconds.
- The system requires that buffer for another I/O transfer.
- A user executes the sync command.

For further information, read sync(1M), sync(2), and update(8) in the ULTRIX-11 Programmer's Manual, Volume 1.

Although it improves throughput, block (buffered) I/O can result in file system inconsistencies. For example, when the system halts unexpectedly (crashes), the blocks that remain in the buffer cache may not be completely written out to disk. When this occurs, the information the system uses to define and access the file system may not be consistent with the file system's current status on disk.

For further information, read Section 4.1.1, Checking File System Consistency, and Section 4.1.2, Correcting File System Inconsistencies.

1.2.16 Character I/O Mode

In character (raw) I/O mode, the ULTRIX-11 kernel transfers data directly from the device to the user-level process's buffer. Character I/O transfers begin on a block boundary and have a length that is a multiple of 512 bytes. Normally, the ULTRIX-11 kernel uses character I/O mode to transfer large amounts of data directly to or from devices without concern for any file structure. For example, character I/O mode is used for swapping and for file system maintenance.

1.2.17 Major/Minor Device Numbers

When accessing a configured device, the ULTRIX-11 system determines the initial mapping information in the same way that it does for a file or directory. In this case, the system searches the /dev directory for an entry with the designated name and uses the listed inumber to access the appropriate inode.

Once reading the appropriate inode, the ULTRIX-11 kernel uses the major and minor device numbers that are listed in place of block addresses to activate that device. The major number specifies the device controller and associates the controller with the appropriate software device driver. Within the ULTRIX-11 system, devices are classified either as block-mode devices (disks or tapes) or as character-mode devices (line printers and communications multiplexers).

Block-mode devices (disks and tapes) also are configured as character-mode devices. They, therefore, have two entries in the /dev directory: one for block-mode access and one for character-mode access. By contrast, character-mode devices have only one entry in the /dev directory.

For disks, the minor number indicates the device unit number and specifies other device-dependent information. For small disks, the minor number indicates the physical unit number. For large disks, the minor number indicates the disk partition, unit number, and, in the case of MSCP disks, the controller number:

- Bits 0-2 indicate the partition
- Bits 3-5 indicate the unit number
- Bits 6-7 indicate the MSCP controller number (0-2)

NOTE

The ULTRIX-11 software divides the large disks into eight partitions and accesses each partition as a pseudodisk. The size and location of the pseudodisks are determined by a sizes table in the device driver. For further information, read Section 1.3, Logical Partitioning of Disks, and Appendix D, Disk Logical Partition Sizes.

For tapes, how the minor number is used is device dependent. TE16, TU16, and TU77 are dual density tapes and use the minor number as follows:

- Bits 0-5 indicate the unit number
- Bit 6 indicates the density: 0 (1600 bpi) or 1 (800 bpi)
- Bit 7 indicates no rewind on close

TE10, TS03, and TU10 are single density tapes (800 bpi) and use the minor number as follows:

- Bits 0-2 indicate the unit number
- Bits 3-6 are unused
- Bit 7 indicates no rewind on close

TK25, TS11, TSV05, and TU80 are single density tapes (1600 bpi) and use the minor number as follows:

- Bits 0-6 are unused (unit 0 only)
- Bit 7 indicates no rewind

The no-rewind feature lets files be appended to the end of a

previously written tape.

For a list of ULTRIX-11 devices and their major device numbers, read Appendix C, ULTRIX-11 Device Names and Major Device Numbers.

1.3 Logical Partitioning of Disks

The ULTRIX-11 system accesses partitioned or nonpartitioned disks:

Partitioned	Nonpartitioned
RK06/7	RX50
RP02/3	RX02
RM02/3	RK05
RM05	ML11
RP04/5/6	
RA60	
RA80/RA81	
RC25	
RL01/2	
RD51/RD52	

For nonpartitioned disks, the system accesses each unit as a single file system. For the larger disks, each physical unit is partitioned into eight logical subunits called pseudodisks. The ULTRIX-11 kernel accesses each pseudodisk as a separate logical file system. For larger disks, therefore, each physical unit can contain multiple file systems. The sizes table in each disk driver determines the size and location of each disk partition.

A typical partition scheme allocates three small partitions for system use, one or more large partitions for user file systems, and one partition to let the entire disk be accessed as a single logical file system. For further information about disk partitions, read Appendix D, Disk Logical Partition Sizes.

NOTE

Logical disk partitions may overlap. As system manager, you must be sure not to place file systems on overlapping partitions. For further information on RA60/RA80/RA81, RC25, and RD51/RD52 disk partitions, read rasize(1M) in the ULTRIX-11 Programmer's Manual, Volume 1.

1.4 Logical Device Names and Special Files

Although a user accesses a device by its special file name, the ULTRIX-11 kernel activates that device by its major and minor device numbers. The special file maps a logical name to an inumber which the system uses to access the appropriate inode and determine the major/minor device numbers and the I/O mode.

The ULTRIX-11 kernel uses either block (buffered) or character (raw) mode special files. Because all block-mode devices usually are also configured as character-mode devices, you can access disks and tapes with either block I/O mode or character I/O mode. By contrast, since they are configured as character-mode special files, you can access line printers and communications devices only with character I/O mode.

To summarize, each special file has these attributes:

- A logical device name
- Major and minor device numbers
- Block or character I/O mode

NOTE

To list the attributes of the configured special files, use the ls command with the -l option specified on the /dev directory.

1.4.1 Creating Special Files

To create a special file, use one of the following commands:

/etc/csf — Use this command to create or remove all special files for a physical unit on a device. For further information, read csf(1M) in the ULTRIX-11 Programmer's Manual, Volume 1.

/etc/mknod — Use this command to create a single special file. For further information, read mknod(1M) in the ULTRIX-11 Programmer's Manual, Volume 1.

/dev/msf — Use this shell script to create or remove special files when specifying a generic device name and unit number. DIGITAL recommends that you use this script to create special files. For further information, read Section 2.6, Updating the Device Support Files.

When creating special files, you can assign arbitrary

logical device names. To provide a consistent I/O interface, however, you should use the ULTRIX-11 device naming convention. When you use the msf script and specify a generic device name, the system automatically assigns the appropriate ULTRIX-11 mnemonic to the device special file. For a list of ULTRIX-11 device mnemonics, read Appendix C, ULTRIX-11 Device Names and Major Device Numbers.

The next five sections discuss the ULTRIX-11 special file names assigned for:

- Partitioned disks
- Nonpartitioned disks
- Magnetic tape
- Terminals
- Miscellaneous devices

1.4.2 Partitioned Disks

For the large disks, each physical unit is partitioned into a maximum eight logical subunits. For partitioned disks, special file names are assigned in the format:

```
/dev/xxnp     (block I/O)
/dev/rxxnp    (character I/O)
```

xx Indicates the ULTRIX-11 device mnemonic.

r Indicates a character (raw) mode special file.

n Indicates the device unit number.

p Indicates the partition number (0-7).

For further information, read Section 2.6.1, Support Files for Disks.

The following are examples of a block and character special file that both access an RP02/3, unit 1, partition 0:

```
/dev/rp10
/dev/rrp10
```

NOTE

You can access RA60, RA80, RA81, RX50, RC25, RD51, and RD52 disks with the same software driver. The RA, RC, and RD disks follow the partitioned disk naming convention for special files. The RX50 disk uses the nonpartitioned disk special file names. The msf command accommodates this inconsistency.

1.4.3 Nonpartitioned Disks

For smaller disks, each physical unit is accessed as one file system. For nonpartitioned disks, special file names are assigned in the format:

```
/dev/xxn    (block I/O)
/dev/rxxn   (character I/O)
```

xx Indicates the ULTRIX-11 device mnemonic.

r Indicates a character (raw) mode special file.

n Indicates the device unit number.

The following are examples of a block and character mode special file that both access an RK05, unit 0:

```
/dev/rk0
/dev/rrk0
```

1.4.4 Magnetic Tapes

For tape, special file names are assigned by density and unit number in the format:

800 bpi	1600 bpi	
/dev/mt#	/dev/ht#	(block I/O)
/dev/rmt#	/dev/rht#	(character I/O)
/dev/nrmt#	/dev/nrht#	(character I/O)

mt Indicates 800 bpi.

ht Indicates 1600 bpi.

# Indicates the device unit number.

r Indicates character (raw) I/O mode.

n Indicates no rewind when the special file is closed.

NOTE

The TM11 operates at 800 bpi and uses the mt mnemonic only. The TS11, TSV05, TU80, and TK25 are 1600 bpi and use the ht mnemonic only. The TM02/3 is dual density and uses both the mt and ht mnemonic. For example, /dev/rmt0 accesses TM02/3, unit 0 at 800 bpi, while /dev/rht0 accesses TM02/3, unit 0 at 1600 bpi.

1.4.5 Terminals

For terminals, special file names are assigned for each communication interface in the format:

/dev/tty##

tty Indicates the terminal mnemonic.

Indicates the 2-digit terminal number.

Once created, this special file associates a terminal with a port on either a multiline interface (for example, DH11) or a single line interface (for example, DL11). A user-level process then can access the special file /dev/tty## to access the control terminal. The file /dev/console refers to the console terminal.

For further information about general terminal interfaces, read tty(4) in the ULTRIX-11 Programmer's Manual, Volume 1. For a description of how tty special files are created, read Section 2.6.3, Support Files for TTY Interfaces.

1.4.6 Miscellaneous Devices

To create special files for the miscellaneous devices, use the csf or msf command. The following lists the miscellaneous special files:

/dev/errlog	Refers to the error log area on the system disk. The ULTRIX-11 error logger creates and uses this special file. For further information, read Chapter 8, ULTRIX-11 Error Logger.
/dev/kmem	Refers to kernel mapped memory. You can use this special file to access kernel memory.
/dev/lp	Refers to the main line printer. The lpr command uses this special file.
/dev/mem	Refers to physical memory. You can use this special file to access physical memory.
/dev/null	Refers to the system null file. When you write to this special file, your data is discarded. When you read this special file, you immediately receive an end-of-file.
/dev/swap	Refers to the swap area on the system disk. This file lets system commands (for example, ps) access the swap area without having to know where it is.

For further information about these special files, read Section 4, Special Files, in the ULTRIX-11 Programmer's Manual, Volume 1.

1.5 Disk Unit Numbers

For most disks, the unit number logical assignment plug (LAP) specifies the device unit number. The following are guidelines for dealing with disk unit number assignment plugs.

- Before booting the operating system, determine the desired unit number for each disk and insert the correct LAP into the disk unit.
- Before changing a disk unit number, make sure all file systems on that disk are unmounted.
- For the RL01 and RL02 disks, the first access of the RL11 controller determines the status of each unit. Subsequent changes cannot be detected. If you turn off the power from a unit or remove the LAP once the status of a unit is determined, that unit is considered nonexistent. Interchanging unit numbers between an RL01 and RL02 disk causes the driver to miscalculate the size of each disk. When using RL01/02 disks, follow the first guideline carefully.
- Disk unit number assignments do not have to be contiguous. The number of units that you specify at sysgen time must be one greater than the highest unit number assigned. For example, if you assign unit numbers 0, 2, and 3, you then should configure the system for four disk units.
- Do not duplicate unit numbers for dual access disks. For example, four disks connected to one controller through port A and to another controller through port B must each have a unique unit number.

NOTE

DIGITAL recommends that you not change a disk's unit number while the operating system is running.

1.5.1 RD51/RD52/RX50 Unit Numbers

The RD51 and RD52 Winchester disks and the RX50 diskettes do not use unit number select plugs. The RQDX1 controller determines the unit number of each disk during initialization. The ULTRIX-11 system supports two Micro/PDP-11 disk configurations. The basic configuration uses a single Winchester disk as unit 0 and the RX50 disks as units 1 and 2, respectively. The alternate configuration uses a second Winchester disk as unit 1 and changes the RX50 unit numbers

to 2 and 3.

Because unit number plugs are not used, the physical location of each unit determines the Winchester and RX50 unit numbers. A Winchester disk (unit 0) is always mounted in the basic Micro/PDP-11 package. A second Winchester disk (unit 1) is mounted external to the basic package.

The Micro/PDP-11 may be either rack mounted or packaged as a free-standing unit. When rack mounted, the Micro/PDP-11 is horizontal, and the lower numbered RX50 unit is the top drive. When free standing, the Micro/PDP-11 is vertical, and the lower numbered RX50 unit is on the left.

Chapter 2

ULTRIX-11 System Generation

During an ULTRIX-11 system generation, you tailor the ULTRIX-11 kernel to your:

- Processor type
- System disk
- Peripheral devices
- Amount and type of work to be done

You may generate either a split I and D kernel, for those processors that have separate Instruction and Data space, or a nonsplit I and D kernel, for those that do not. For a detailed description of the kernel and processor types that have split Instruction and Data space, read the ULTRIX-11 Software Technical Description.

You also tailor the ULTRIX-11 kernel for the type of system disk and peripheral devices used. Then, by adjusting the values of the system parameters, you tailor the ULTRIX-11 kernel to the number of users it is to support. These system parameters determine the process table sizes and various other internal data structures.

The remaining sections of this chapter discuss the generation process and the fatal sysgen error messages. An ULTRIX-11 system generation consists of:

- Preparing for system generation
- Creating the system configuration file
- Making the ULTRIX-11 kernel
- Installing the ULTRIX-11 kernel
- Updating the device support files

For a sample system generation, read Appendix A, Sysgen Program Example.

2.1 Preparing for System Generation

As preparation for an ULTRIX-11 system generation, you should familiarize yourself with the sysgen program. In addition, you should gather all the pertinent system hardware information.

2.1.1 Running the sysgen Program

To run the sysgen program, you first must log in to the sys account. Once you have logged in, the system prints the following message and the /bin/sh prompt:

```
Welcome to the ULTRIX-11 System

$
```

Then, use the cd command to change directories to /sys/conf. In response to the shell prompt, type this command sequence:

```
$ cd conf
```

Once in the /sys/conf directory, you are ready to run the sysgen program.

To start the sysgen program running, type:

```
$ sysgen
```

Once loaded and running, the sysgen program prints the following messages and a sysgen> prompt:

```
ULTRIX-11 System Generation Program

For help, type h then press <RETURN>

sysgen>
```

The sysgen> prompt indicates that the program is ready to accept your commands.

The sysgen program has an extensive on-line help facility. The help information is the main source of documentation for the system generation process. For help information, type:

```
sysgen> h
```

The sysgen program then prints:

The "sysgen>" prompt indicates the sysgen program is ready to accept commands. To execute a command you type the first letter of the command, then press <RETURN>. Some of the commands will ask you for additional information, such as a file name. For more help with a command, type h followed by the command letter then press <RETURN>. For example, "h c" for the create command.

Command	Description
<CTRL/D>	Exit from sysgen (backup one question in "c" command).
<CTRL/C>	Cancel current command, return to "sysgen>" prompt.
!command	Escape to the shell and execute "command".
[c]reate	Create an ULTRIX-11 kernel configuration file.
[r]emove	Remove an existing configuration file.
[l]ist	List names of all existing configuration files.
[p]rint	Print a configuration file.
[m]ake	Make the ULTRIX-11 kernel.
[i]nstall	Print instructions for installing the new kernel.
[d]evice	List configurable processors and peripherals.
[s]ource	Compile and archive a source code module (u1.c, etc.).

The sysgen sequence is: use "c" to create the configuration file, "m" to make the new kernel, and "i" for installation instructions.

2.1.2 Gathering System Information

You also should have all the relevant documentation readily available. DIGITAL recommends that you have this documentation available for use:

- System hardware documentation
- ULTRIX-11 Installation Guide
- ULTRIX-11 Software Technical Description

Then, you should gather the system information that is needed to generate your ULTRIX-11 kernel. While doing so, consider the following and write the information on the system configuration worksheet, Appendix G in the ULTRIX-11 Installation Guide:

- Determine your processor type. The most significant consideration is whether the processor has separate Instruction and Data space. The ULTRIX-11 software supports all memory-managed PDP-11 processors from the PDP-11/23 through the PDP-11/73. The ULTRIX-11 software, however, does not support the LSI-11/03 and the Personal Computer series. The Micro/PDP-11 uses the PDP-11/23 or PDP-11/73 processor. If the processor is not listed as supported by the ULTRIX-11 software technical description, use the processor type with features that most closely match those of the processor you are using. For example, the PDP-11/35 uses processor type 40, the PDP-11/50 uses processor type 45 or 55.

- Because the sysgen program assigns disks by controller type, determine the type of each disk controller present and the drives that are connected to it. To display a list of ULTRIX-11 supported controllers and drives, use the sysgen device command.

- Determine the type of each tape controller and tape drive connected to it.

- Determine the type of each communications interface and the number of units for each interface type.

- Determine what peripherals are to be configured into the system.

- If the system is to support any user-written device drivers, read Section 2.8, Sysgen of User-Written Device Drivers.

- Determine the hardware CSR address and interrupt vector address of all devices that are to be configured into the system. Most devices, except communication interfaces, have a standard address and vector. The sysgen program knows the standard address and vector for all supported devices.

- Determine the disk controller and drive where the ULTRIX-11 root and swap file systems and error log file are to reside. The sysgen program has standard locations for these files. The program can also display a help message to give guidelines for changing their placements.

- According to the number of users the ULTRIX-11 system is to support, determine the values of the system parameters. The sysgen program has a standard value for each parameter. Each standard value is based on an estimated number of users for each processor type. For guidelines for changing parameter values, read the help message text in Appendix A, Sysgen Program Example.

2.2 Creating the System Configuration File

To create the system configuration file, use the sysgen program and specify the create command. In response to the sysgen> prompt, type:

```
sysgen> c
```

Once the create command begins executing, the sysgen program first prompts you for a configuration file name. To use the default name (unix), press the <RETURN> key. Then, it prompts you for information about your system and writes your responses to this configuration file. You later use this configuration file to make your ULTRIX-11 kernel.

You should respond to each sysgen prompt with the appropriate information gathered earlier (Section 2.1.2). For each prompt, the sysgen program prints either a default response or a list of possible responses. To use the default response to any given prompt, press the <RETURN> key. If you need further help with any given prompt, type ? and press the <RETURN> key.

If you have made a mistake when entering information to a given prompt, press <CTRL/D>. The <CTRL/D> erases all the information entered for that prompt and backs up to the previous prompt. You can press <CTRL/D> repeatedly to back up more than one prompt. As you do so, however, you erase the information for each prompt.

2.3 Verifying Your Responses

When you have answered all sysgen prompts, you should verify your responses. To verify the newly created configuration file, use the sysgen program and specify the print command. In response to the sysgen> prompt, type:

```
sysgen> p
```

This configuration file contains the specifications that define the necessary information about the system hardware configuration and parameter values.

The /sys/conf directory contains several prototype configuration files. You can use the sysgen program to:

- List their file names
- Print their contents
- Remove an unwanted file

2.4 Making the ULTRIX-11 Kernel

To make the ULTRIX-11 kernel, use the sysgen program and specify the make command. In response to the sysgen> prompt, type:

```
sysgen> m
```

Once the make command begins executing, the sysgen program prompts you for the configuration file name. If you used the default configuration file name (unix), you should press the <RETURN> key. If you used a different configuration file name, however, you should type that name.

The ULTRIX-11 kernel is produced in two phases from ten files or archives:

c.c	Configuration tables
dds.c	MSCP data structures
dds.h	Disk driver headers
dump.s	Crash dump code
ec.c	Additional configuration tables
l.s	Locore vectors
mch0.s	Machine and dump header file
mch.o	Machine language assist
LIB1	System library
LIB2	Device driver library

First, the sysgen program calls the mkconf program. The mkconf program reads the specified configuration file and creates the c.c, dds.h, l.s, and mch0.s files. In addition, it prints a description of the system configuration. For further information, read mkconf(1M) in the ULTRIX-11 Programmer's Manual, Volume 1.

Second, the sysgen program executes the makefile in the /sys/conf directory. This makefile first assembles the l.s and dump.s files and compiles the c.c, dds.c, and ec.c files. Then, to link the object files and libraries, the makefile calls the link editor (ld). For further information, read make(1) in the ULTRIX-11 Programmer's Manual, Volume 1.

As indicated, the sysgen program itself does not make the kernel. Rather, it calls other programs that do the actual work. When making the ULTRIX-11 kernel, these programs display a series of informational messages. The final message indicates whether the sysgen was successful. For sample messages, read Appendix A, Sysgen Program Example.

2.5 Installing the ULTRIX-11 Kernel

To install the ULTRIX-11 kernel, use the sysgen program and specify the install command. In response to the sysgen> prompt, type:

```
sysgen> i
```

Once the install command begins running, the sysgen program prints the procedure for installing the new ULTRIX-11 kernel:

Use the following procedure to install the new kernel:

- Type <CTRL/D> to exit from the sysgen program.
- Become superuser (type "su", then enter the root password).
- Move the new kernel to the root (mv unix.os /nunix).
- Type <CTRL/D> twice (to logout), then login to the operator account. Use the operator services "s" command to shutdown the system and "halt" command to halt the processor.
- Use the manual boot procedure, described in section 3.4 of the ULTRIX-11 System Management Guide, to boot the new kernel. For example, "Boot: rl(0,0)nunix" from an RL02 disk.
- Save the old kernel then rename the new kernel "unix". (mv unix ounix; mv nunix unix; chmod 644 unix)
- Set the date (date command), check the file systems (fsck), then type <CTRL/D> to enter multi-user mode.

When an ULTRIX-11 kernel is made, the system creates an executable file in the /sys/conf directory. This file usually is given the name of the configuration file that was used when making the kernel plus the .os suffix. For example, if you used the default configuration file name earlier, the ULTRIX-11 kernel is named unix.os. Because, during normal operations, several system-level commands look for the file /unix, your current, running kernel should always be named unix.

The specific details of a kernel installation are hardware dependent. For further information, read Section 1.6 in the ULTRIX-11 Installation Guide.

2.6 Updating the Device Support Files

As the final step in the system generation process, you may have to update the device support files. The device support files let the operating system and users access devices.

During your ULTRIX-11 software installation, you created the device support files for all the devices on your system. Unless you have installed new devices or changed existing ones, these files do not need to be updated. If you have installed additional disks or tapes, however, you must create the device support files for those devices only. If you have reassigned the terminal numbers (tty##), you should update the TTY support files for those devices only.

The next five sections discuss the support files that you must update for each device type. For information about device support files and their relation to the various classes of devices, read Chapter 1, ULTRIX-11 I/O System, and Chapter 4, ULTRIX-11 Maintenance and Administrative Functions.

2.6.1 Support Files for Disks

A set of special files must exist in the /dev directory for each configured disk unit. For further information, read Section 1.4, Logical Device Names and Special Files.

To create special files for disks, become the superuser (su command) and change your directory to /dev. Once in the /dev directory, use the msf script. For disks that are not connected to a MASSBUS or MSCP disk controller, type this command sequence:

```
# msf xx n
```

xx Specifies the generic disk name.

n Specifies the unit number.

For example, to create the special files for an RK07, unit 3, type:

```
# msf rk07 3
```

MASSBUS disks (RM02, RM03, RM05, RP04, RP05, RP06, and ML11) can be connected to any one of three RH11/RH70 disk controllers. For disks that are connected to a MASSBUS disk controller, type this command sequence:

```
# msf xx_c n
```

xx_c Specifies the generic disk name (xx) and the RH11/RH70 controller number that it is connected to. The RH11/RH70 controller numbers are determined by the order in which they are entered in the sysgen configuration file. Specifically, the first entered is numbered 0, the second 1, and the third 2. If the disk is connected to the first RH11/RH70 controller, the controller number (_c) may be omitted.

n Specifies the unit number.

For example, this command sequence creates the special files for an RP06, unit 0 on the first RH11/RH70 controller:

```
# msf rp06 0
```

For example, this command sequence creates the special files for an RM03, unit 2 on the second RH11/RH70 controller:

```
# msf rm03_1 2
```

If the system disk is connected to a MASSBUS controller, you must assign it to the first RH11/RH70 controller. Otherwise, you can assign MASSBUS disks arbitrarily.

During a sysgen, you also can change the CSR and vector addresses of each MASSBUS controller. By convention, the default CSR and vector addresses for the RH11/RH70 controllers are:

RH #	Name	CSR	Vector
first	HP	176700	254
second	HM	176300	150
third	HJ	176400	204

NOTE

When you use the msf command to create the special files for ML11 disks, four special files are created for each disk. Specifically, a block and raw (character) special file is created for each controller type (hp## and rhp##, hm## and rhm##, or hj## and rhj##). In addition, a block and character special file with the names ml# and rml# are created. These files are links to the appropriate controller special files.

You can configure your ULTRIX-11 kernel with up to three MSCP disk controllers, but you must configure only one of

each type. The supported MSCP controllers and disks are:

```
UDA50 - RA60/RA80/RA81
KLESI - RC25
RQDX1 - RX50/RD51/RD52
RUX1  - RX50
```

For disks that are connected to a MSCP disk controller, type this command sequence:

```
# msf xx_c n
```

xx_c Specifies the generic disk name (xx) and the MSCP controller number that it is connected to. The MSCP controller numbers are determined by the order in which they are entered in the sysgen configuration file. Specifically, the first entered is numbered 0, the second 1, and the third 2. If the disk is connected to the first RH11/RH70 controller, the controller number (_c) may be omitted.

n Specifies the unit number.

For example, this command sequence creates the special files for an RA60, unit 1 on the first MSCP controller:

```
# msf ra60 1
```

For example, this command sequence creates the special files for an RC25, unit 0 on the second MSCP controller:

```
# msf rc25_1 0
```

The MSCP disk controllers are assigned the same default CSR and vector address. During a sysgen, you must assign alternate CSR and vector addresses to the second and third controller (if present).

Because several system commands access /etc/fstab for their default argument lists, you also must update /etc/fstab after creating special files for disks. The /etc/fstab file associates each mountable file system with the directory on which you mount it. The system commands that access /etc/fstab include the df, fsck, mount, quot, and umount commands. For further information, read fstab(5) in the ULTRIX-11 Programmer's Manual, Volume 1 and Section 4.6.7, Mounting and Unmounting File Systems.

2.6.2 Support Files for Magnetic Tape

Like disks, each tape unit requires a set of special files in the /dev directory. Although some commands (for example, dump and restor) have default special file names built in,

tape units have no other support files. For further information, read Section 1.4, Logical Device Names and Special Files.

To create special files for tapes, become the superuser (su command) and change directory to /dev. Once in the /dev directory, use the msf script. In response to the superuser prompt, type this command sequence:

```
# msf xx n
```

xx Specifies the generic tape unit name.

n Specifies the unit number.

For example, to create a special file for a TE16, unit zero, type this command sequence:

```
# msf te16 0
```

2.6.3 Support Files for TTY Interfaces

Unlike disks and tapes, each terminal requires only one special file in the /dev directory. For further information, read Section 1.4, Logical Device Names and Special Files.

To create a special file for terminals, become the superuser (su command) and change directory to /dev. Once in the /dev directory, use the msf script. In response to the superuser prompt, type this command sequence:

```
# msf xx n tty##
```

xx Specifies the generic device name.

n For KL11 and DL11 interfaces, specifies the number of units. For all other communications interfaces, specifies the unit number.

tty## For the KL11 and DL11, associates the unit with the name of the special file for its terminal. For the multiline interfaces, specifies the terminal special file name associated with the first line on the unit. The remaining lines in the multiline interface are assigned sequentially (for example, tty##+1, tty##+2).

For example, the following sequence of commands creates a series of TTY special files. The make command with the ttyclean argument specified removes all existing TTY special files:

```
# cd /dev
# make ttyclean
# msf dz11 0 tty00
# msf dz11 1 tty08
# msf dh11 0 tty16
# msf dl11 3 tty32
```

When making KL11 or DL11 special files, count the total number of KL11 devices (excluding the console) and DL11 devices. Then, create that number of special files with the msf command. For example, if you have a single KL11 and two DL11 devices, the following command sequence assigns the KL11 to tty32 and the DL11s to tty33 and tty34:

```
# msf dl11 3 tty32
```

Having created a TTY special file for each terminal, you also must create the initialization data for each. More specifically, you must set the local/remote and terminal characteristics that the system accesses from the /etc/ttys file. For further information, read Section 4.7.2, Creating Communications Interface Special Files, and tty(5) in the ULTRIX-11 Programmer's Manual, Volume 1.

2.6.4 Support Files for Miscellaneous Devices

Like terminals, miscellaneous devices normally require only one special file in the /dev directory. For further information, read Section 1.4, Logical Device Names and Special Files.

To create a special file for miscellaneous devices, become the superuser (su command) and change directory to /dev. Once in the /dev directory, use the msf script. In response to the superuser prompt, type this command sequence:

```
# msf xx n
```

xx Specifies the generic device name.

n Specifies the unit number.

When ULTRIX-11 does not support multiple units for a specified device, the unit number is ignored:

```
# msf lp11 0
# msf dn11 1
# msf du11 0
```

For a detailed description of the miscellaneous special files, read Section 1.4.6, Miscellaneous Devices, and

Section 4, Special Files, of the ULTRIX-11 Programmer's Manual, Volume 1.

2.6.5 The /dev/swap File

The /dev/swap special file refers to the file system or disk partition on which the swap area resides. By accessing /dev/swap, some programs (for example, ps) can access the swap area without having to know the name of the swap device.

If you generate the system with the swap area in the standard place, you do not need to create or modify the /dev/swap file. If you use a nonstandard placement of the swap area, however, the sysgen program automatically prints a warning message to indicate that you must modify the /dev/swap file.

To modify the /dev/swap file, become the superuser (su command) and change to the /dev directory. Once in the /dev directory, use the msf script to create the special file for the disk partition where the new swap area is to reside. Then, remove the current swap file and link the special file for the swap partition to /dev/swap. Finally, change to the root directory (/) and use the sync command.

The following example lists the sequence of commands for placing the /dev/swap file on a file system on an ML11, unit 0:

```
# cd /dev
# msf ml11 0
# rm swap
# ln ml0 swap
# cd /
# sync
```

The /dev/swap file is always linked to the block mode special file for the swap file system. Do not use the raw special file. In this example, ml0 is used instead of rml0.

NOTE

You must modify the /dev/swap file when the system is in single-user mode only.

2.7 Error Messages

The sysgen program has extensive error checking and can generate a large number of messages. Usually, the majority of these messages are self-explanatory and cover basic errors. The sysgen program checks for and reports when you have:

- Entered too many characters on a line
- Entered an invalid device or file name
- Omitted a required value
- No access permission to a file

In addition, the sysgen program prints messages that indicate that a fatal error has occurred. The next five sections discuss the fatal sysgen error messages.

2.7.1 sysgen: Can't exec fresh copy

After creating a configuration file, the sysgen program uses the exec() system call to overlay itself. This action is necessary to reset its internal tables to a known state. This error message indicates that the exec() system call failed. When the system prints this error message, it also indicates the specific hardware or software problem involved.

2.7.2 Can't make unix from configuration xxxx.os!

The sysgen program actually calls other programs to make the ULTRIX-11 kernel. This error message indicates that one of these programs encountered an error. When creating the ULTRIX-11 kernel, the called programs print progress messages and error messages. These messages normally provide information that is helpful in determining the exact cause of this error.

2.7.3 FATAL size error xxxx.os should not be installed!

During the last phase of a kernel generation, the sysgen program checks the size of the newly generated kernel to ensure that it is within acceptable limits. This message indicates that the named kernel is too large to run in the processor's available address space.

This error can occur when you have attempted to configure too many devices into the system. However, this situation is unlikely for any reasonable hardware configuration. A more likely cause is when you run out of kernel data space due to an inordinate increase in the value of a system parameter. The sysgen program prints the actual kernel size and the size limit. Use these values to determine by how much the kernel size must be decreased. When you reduce a parameter value by one, the help message for the sysgen "Use standard system parameters" prompt prints the number of

bytes you save. For further information, read Appendix A, Sysgen Program Example.

2.7.4 MAPPED BUFFERS - forbidden zone violation!

To increase the amount of available data space, the kernel uses a mapped buffer scheme. This mapped buffer scheme involves moving the I/O buffer cache outside the kernel data space and dynamically remapping between the buffers and the 8K-byte data segment at virtual address 0120000. This imposes the restriction that you cannot access data space symbols in the 0120000 - 0140000 address range (forbidden zone) while memory management segmentation register 5 is mapped to the I/O buffers.

This error message occurs when the value of the mb_end symbol exceeds address 0120000. Usually, the data space arrangement used by the kernel prevents this error from occurring. When this error does occur, you must decrease the size of the data structures with addresses less than the mb_end symbol until the value mb_end is less than address 0120000.

In most cases, you can recover from this error by reducing the number of buffers (NBUF) configured into your system. Each NBUF structure uses 30 bytes of data space. To reduce the number of configured buffers, subtract 120000 (octal) from the value of mb_end and convert the result to decimal. Then, divide the result by 30 and add one. Finally, you must reconfigure your system and reduce the number of buffers to this calculated value.

For example, the following illustrates using the dc command to calculate a new NBUF value when _mb_end is 121234:

```
$ dc
8i
121234
120000
-p
668
q
$ dc
668
30
/p
22
1
+p
23
q
$
```

Having calculated a new NBUF value, you then must use the

sysgen program with the c command specified to create a new configuration file. Finally, you must remake and install this kernel.

If this procedure does not reduce the _mb_end sufficiently, you must determine which additional data structure to decrease. To do so, use the sysgen "standard parameters" help text. Not all standard parameters, however, affect the value of the mb_end symbol. The ULTRIX-11 software allows the data symbols which are not accessed while the I/O buffers are mapped in (NCALL, NFREE, NPROC, and NTEXT) to be assigned beyond the mb_end symbol.

2.7.5 UNIBUS MAP - forbidden zone violation!

On PDP-11/24s with KT24 memory expansion, PDP-11/44s, and PDP-11/70s, the UNIBUS map uses 31 registers to map the 18-bit UNIBUS addresses to the 22-bit addresses needed to access memory. Each register maps 8K-bytes of memory. On these processors, all Direct Memory Access (DMA) transfers must go through the UNIBUS map.

DMA access to data structures in the ULTRIX-11 kernel is required by several hardware devices to pass control information between the device driver and the device. These kernel data structures are:

- Disk bad block buffers
- MSCP disk packet buffers
- TS11 tape packet buffers
- DH11 clists
- User-written device driver buffers

The first UNIBUS register is permanently allocated to mapping DMA transfers to and from these kernel data structures. Assigning only one register limits the size of these structures to 8K-bytes.

During the last phase of making a new kernel, the system checks the total size of these data structures. If this size exceeds 8K-bytes, it prints the UNIBUS MAP warning message.

Usually, this size limit is large enough to accommodate most reasonable configurations. If this error does occur, however, you must reduce the size of these data structures. More specifically, you must decrease the number of devices, the size of the clists, or the size of the data structures in any user-written drivers.

2.8 Sysgen of User-Written Device Drivers

For special purpose user devices, the ULTRIX-11 operating system lets you incorporate user-written software drivers into the kernel. This section describes the procedure that you should follow when generating a kernel that is to include user-written device drivers. It is assumed that you are a system programmer and are familiar not only with writing device drivers but also with interfacing them to the UNIX operating system.

The ULTRIX-11 system is shipped with four user-modifiable device driver source code modules in the /sys/dev directory: u1.c, u2.c, u3.c, and u4.c. Correspondingly, the ULTRIX-11 device mnemonics for the user-written devices are u1, u2, u3, and u4. Each prototype user device driver contains null functions and dummy data structure definitions to satisfy all possible interface requirements to the ULTRIX-11 I/O system (block or character mode). For information about writing a device driver and interfacing it into the ULTRIX-11 kernel, read Appendix H, ULTRIX-11 User Device Driver Commentary.

To create a user-written device driver, edit one of the prototype source modules. Specifically, edit the driver source code into the null functions as needed. You should not delete the unused null functions. Then, replace the dummy definitions with actual data structures (for example, device register definitions). Finally, to configure a user-written device, use this procedure:

1. Study one of the prototype user device drivers to determine which functions and data structure definitions you need to replace.

2. Copy the unmodified prototype driver to a backup file.

3. Edit the prototype source module and enter the user-written device driver source code.

4. Run the sysgen program and use the s command to compile and archive the user device driver.

5. Use the sysgen program to create a new configuration file that includes the user-written driver. At the appropriate prompt, type the names of the user-written device drivers (u1, u2, u3, or u4).

6. Use the sysgen program to make a new ULTRIX-11 kernel.

7. Install and boot the new kernel.

8. Use the mknod command to create the user device special files. The file /usr/include/sys/devmaj.h contains the

major device numbers reserved for user devices.

NOTE

DIGITAL has verified the above procedure by editing the code from the TS11 tape driver and the DZ11 communications interface driver into prototype modules u1.c and u2.c. DIGITAL also generated an ULTRIX-11 kernel with u1 and u2 configured and verified that user devices u1 and u2 function as expected.

When writing device drivers for user devices, you should consider:

- The system configuration file specifies the device's hardware device register address and interrupt vector address in the same manner as standard devices. The u1start() function in the u1.c prototype module contains an example of the procedure for obtaining the device's hardware register address. The driver does not need to know the device's interrupt vector address. If the device interrupts at a bus request level other than BR 5, however, you must edit the /sys/conf/l.s file to change the bus request level. If you change the l.s file, you cannot use the sysgen make command. In response to the superuser prompt, type this sequence of commands:

```
# cd /sys/conf
# mkconf <unix.cf
# make unixnn
```

  nn Specifies the processor type (for example, 23, 44, 45).

- You cannot interface user devices with the ULTRIX-11 error logging system. To report errors, you should use the deverror() and printf() routines.

- When the processor has a UNIBUS map, the strategy function must call the UNIBUS map allocation function mapalloc(bp). The mapalloc(bp) function requires as an argument a pointer to an I/O buffer header. The system releases the UNIBUS map registers after completing the physical I/O transfer.

- When the device uses a bus address extension register for 22-bit addressing, the driver must load the full bus address into the BAE register and not use the UNIBUS map.

Chapter 3

ULTRIX-11 Boot Procedures

The ULTRIX-11 operating system usually resides on the system disk in the /unix file. Booting the ULTRIX-11 system involves a primary and a secondary boot program. These boot programs load the /unix file into memory, initialize it, and start it running in single-user mode.

When you use the autoboot procedure, the boot programs load the operating system and start it running without intervention. When circumstances require you to boot the system from an alternate device or to boot one of the stand-alone programs, however, you can use a manual boot procedure.

The remaining sections of this chapter discuss:

- Boot sequence
- Boot file specifications
- System disk autoboot procedure
- System disk manual boot procedure
- Magnetic tape manual boot procedure
- RX50 manual boot procedure
- Boot options
- Boot error messages

3.1 The Boot Sequence

This section describes the ULTRIX-11 boot sequence. This information is intended for reference only and is not required for booting the ULTRIX-11 operating system.

The hardware boot ROM loads the primary boot program from block 0 of the boot device into memory (physical location 0). Once the primary boot program is loaded, the hardware boot ROM starts it by jumping to location 0.

If it could not load the primary boot program, the hardware boot ROM prints an error message or loops endlessly. (The newer boot ROMs have error recovery, while most of the older boot ROMs do not.)

If loaded from disk, the primary boot program relocates itself to address 0157000 in high memory and begins execution. If loaded from tape, however, it relocates itself to address 137000 and begins execution.

Once the primary boot program begins execution, it loads the secondary boot program from the boot device. If the boot device is a disk, the secondary boot is loaded from the /boot file (root file system). When the secondary boot program is loaded, the primary boot starts it running by a jump to location 0.

If the /boot file could not be loaded, the secondary boot is loaded from the /boot.bu file (root file system). If neither file can be loaded, the primary boot program loops endlessly.

The secondary boot program first sizes memory, then relocates itself to high memory at address 0400000 (128 KB), and starts executing in user mode. If the boot device is the system disk, the secondary boot program automatically loads the operating system, prints this message, and starts the autoboot procedure:

```
Sizing Memory...

Boot: xx(0,0)unix     (<CTRL/C> will abort auto-boot)
```

xx Specifies the ULTRIX-11 device mnemonic for the system disk.

If the boot device is an alternate device, or if you abort the autoboot procedure, the secondary boot program prints this message and a boot prompt:

```
To list options, type help then press <RETURN>

Boot:
```

The boot prompt lets you manually boot the operating system from a file other than /unix or load one of the distributed stand-alone programs. You should respond with the boot file specification. This information tells the secondary boot program the location of the ULTRIX-11 kernel or the stand-alone program. For a description of the boot file specification, read Section 3.2, Specifying the Boot File. For further information about the distributed stand-alone programs, read Appendices B, D, and F in the ULTRIX-11 Installation Guide.

If the system cannot load the designated file, it prints an error message and another boot prompt. If no errors occur, the system loads the file at physical memory location 0 and starts it with a trap instruction. The trap instruction vectors through location 034. The secondary boot program starts the operating system and all stand-alone programs by vectoring through location 034. The system uses this start-up method because the secondary boot program executes in user mode and can not execute a jump to physical location 0.

Before starting the operating system, the secondary boot program performs most of the operations required to initialize the ULTRIX-11 operating system. When the ULTRIX-11 system starts running, it comes up in single-user mode with the superuser account active at the console.

3.2 Specifying the Boot File

To boot a stand-alone program or the ULTRIX-11 system manually, you must specify the file that is to be loaded. In response to the boot prompt, you must specify:

device(unit,offset)file

device — Specifies the 2-character mnemonic of the device on which the file is to be found. For a list of the ULTRIX-11 device mnemonics, read Table 3-1, ULTRIX-11 Device Mnemonics.

unit — Specifies the unit number of the device on which the file is to be found. For ht tape, unit numbers 0-3 specify physical units 0-3 at 800 bpi, while unit numbers 4-7 specify physical units 0-3 at 1600 bpi.

offset — Specifies where, from the beginning of that device, the file is to be found. For disks, the offset is the number of blocks from the beginning of the disk where the appropriate file system starts. For example, on disks, an offset 0 specifies the root file system. For tapes, the offset is the number of files from the beginning of the tape. For example, on tapes, an offset 0 specifies the first file on the tape.

file — Specifies the name of the file that is to be loaded. This name can be either an absolute pathname or a single file name. If file is a file name, the boot program searches the root directory of the file system for that file. You should specify a single file name when booting from tape.

For example, this sequence boots the /unix file from an RL02, unit 0:

```
Boot: rl(0,0)unix
```

The following sequence loads the stand-alone bads program (/sas/bads) from an RK07, unit 0:

```
Boot: hk(0,0)/sas/bads
```

The following sequence loads the stand-alone copy program (/copy) from TM11 tape:

```
Boot: tm(0,0)copy
```

To correct typing mistakes while entering the boot file specifications, press the <DELETE> key to erase a single character and <CTRL/U> to erase the entire line.

Table 3-1 ULTRIX-11 Device Mnemonics

Name	Device Type
hk	RK06/7
hp	RM02/3/5, RP04/5/6, ML11 (first RH11/RH70)
hm	RM02/3/5, RP04/5/6, ML11 (second RH11/RH70)
hj	RM02/3/5, RP04/5/6, ML11 (third RH11/RH70)
ra	RA60/RA80/RA81
rc	RC25
rd	RD51/RD52
rx	RX50
rk	RK05
rl	RL01/2
rp	RP02/3
ht	TU16/TE16/TU77
tm	TU10/TE10/TS03
ts	TS11/TU80/TSV05/TK25

3.3 Autobooting from the System Disk

You normally autoboot the ULTRIX-11 kernel from the root file system on the system disk, unit 0 (for RC25 disks, unit 1). During an autoboot, both the boot programs and the ULTRIX-11 kernel are automatically loaded from the system disk. To autoboot and initialize the ULTRIX-11 kernel, use the following 4-step procedure:

1. Ensure that the processor's HALT switch is released. Then, execute the hardware boot ROM for the system disk. For further information about the boot ROM, read your system's hardware documentation.

   Once loaded and running, the secondary boot program boots the ULTRIX-11 operating system and prints the following messages:

   ```
   Sizing Memory...

   Boot: xx(0,0)unix     (<CTRL/C> will abort auto-boot)

   xx(0,0)unix: size1+size2+size3...sizen
   ```

   xx — Specifies the two character ULTRIX-11 mnemonic for the system disk (for example, hk, hp, ra, rc, rd, rl, or rp).

   size1 — Specifies the size of either the ROOT TEXT segment (nonsplit I and D processor) or the DATA+BSS segment (split I and D processor).

   size2 — Specifies the size of either the DATA+BSS segment (nonsplit I and D processor) or the ROOT TEXT segment (split I and D processor).

   size3 — Specifies the size of the first overlay.

   sizen — Specifies the size of the last overlay. A maximum of seven overlays may be used and their sizes reported.

   Once loaded and initialized, the ULTRIX-11 operating system starts running in single-user mode and prints the following start-up banner:

   ```
   ULTRIX-11 Kernel V#

   realmem = ######
   buffers = ######
   usermem = ######
   erase = delete, kill = ^U, intr = ^C
   #
   ```

V# Specifies the version of the ULTRIX-11 system that was booted.

realmem Specifies the size of all of memory in bytes.

buffers Specifies the amount memory used by the I/O buffer cache.

usermem Specifies the amount of free user memory in bytes. This is the amount of memory available after the ULTRIX-11 operating system and the buffer cache.

erase Specifies that you can use the <DELETE> key to delete by character.

kill Specifies that you can press <CTRL/U> to delete an entire line of input.

intr Specifies that you can press <CTRL/C> to interrupt a program.

NOTE

The system prints the last message to remind you that it has changed these control characters from the standard UNIX V7 characters (where erase = #, kill = @, intr = delete).

Once running and ready to accept commands, the ULTRIX-11 operating system issues a # prompt, indicating that the superuser account is active.

2. Use the date command to set the system date and time. In response to the superuser prompt, enter the following command sequence:

```
# date yymmddhhmm.ss
```

yy Specifies the last two digits of the year (00-99).

mm Specifies the month number (01-12).

dd Specifies the day of the month number (01-31).

hh Specifies the hour of the day (00-23).

mm Specifies the minutes of the hour (00-59).

SS Specifies the seconds of the minute (00-59). This field is optional.

For example, to set the time and date to 6:30 PM on August 28, 1984, type this command sequence:

```
# date 8408281830
```

3. DIGITAL recommends that you check the consistency of your file systems as well as the size and a summary of the error log file after each system boot.

   To check the consistency of your file systems, use the fsck command. In response to the superuser prompt, type this command sequence:

   ```
   # fsck -t /tmp/fsck.temp
   ```

   The -t option specifies the name of a temporary file that is to be used when checking the file systems. For further information, read Section 4.1, File System Maintenance.

   To display the size of the error log file as well as a summary report, use the elp command. In response to the superuser prompt, type this command sequence:

   ```
   # elp -s
   ```

   If the command sequence fails, you may have to initialize the error log file. For further information, read Section 8.1, Operating Procedures.

4. Finally, to make the system available for time-sharing, press <CTRL/D>. This command automatically causes the operating system to change from single-user mode to multiuser mode. The operating system prints a series of messages and a login prompt. For further information, read Section 1.2.4, Multiuser Mode.

3.4 Manually Booting from the System Disk

You normally autoboot the ULTRIX-11 kernel from the root file system on the system disk, unit 0 (for RC25 disks, unit 1). In addition, you can manually boot from the system disk. Specifically, you can load the boot programs from the system disk and then manually boot an ULTRIX-11 kernel or a stand-alone program. For instructions for booting stand-alone programs, read Appendixes B, D, and F in the ULTRIX-11 Installation Guide.

To load the boot programs from the system disk and manually boot the ULTRIX-11 kernel, use the following 3-step procedure:

1. Ensure that the processor's HALT switch is released. Then, execute the hardware boot ROM for the system disk. For further information about the boot ROM, read your system's hardware documentation.

   Once loaded and running, the secondary boot program autoboots the ULTRIX-11 operating system and prints the following messages:

   ```
   Sizing Memory...

   Boot: xx(0,0)unix    (<CTRL/C> will abort auto-boot)
   ```

   xx Specifies the two character ULTRIX-11 mnemonic for the system disk (for example, hk, hp, ra, rc, rd, rl, or rp).

   As soon as the boot prompt appears, however, interrupt the autoboot by pressing <CTRL/C>. When you interrupt the autoboot procedure, the secondary boot program prints this message and a boot prompt:

   ```
   To list options, type help then press <RETURN>

   Boot:
   ```

NOTE

There may be a delay of several seconds between the time that you press <CTRL/C> and the time that the boot prompt reappears. If you wait too long to press <CTRL/C>, the ULTRIX-11 operating system will be loaded into memory and will begin printing several messages. If this happens, you must halt the processor and restart this step.

2. Enter one of the appropriate boot file specifications. For example, the following command sequence boots a test ULTRIX-11 kernel (/unix.test):

```
Boot: rp(0,0)unix.test
```

As it loads the system, the secondary boot program sizes memory and prints this message:

```
rp(0,0)unix.test: size1+size2+size3...sizen
```

size1 Specifies the size of either the ROOT TEXT segment (nonsplit I and D processor) or the DATA+BSS segment (split I and D processor).

size2 Specifies the size of either the DATA+BSS segment (nonsplit I and D processor) or the ROOT TEXT segment (split I and D processor).

size3 Specifies the size of the first overlay.

sizen Specifies the size of the last overlay. A maximum of seven overlays may be used and their sizes reported.

You also can boot other files from other disks by entering the appropriate file specification. For further information, read Section 3.2, Specifying the Boot File.

NOTE

If the operating system that you wish to boot is not named /unix, DIGITAL recommends that you rename it /unix. Several user-level commands and system-level programs use the /unix file for the name list of the operating system that currently is running.

If you manually booted the ULTRIX-11 operating system, it starts running in single-user mode and prints this start-up banner:

```
ULTRIX-11 Kernel V#

realmem = ######
buffers = ######
usermem = ######
erase = delete, kill = ^U, intr = ^C
#
```

V# Specifies the version of the ULTRIX-11 system that was booted.

realmem Specifies the size of all of memory in bytes.

buffers Specifies the amount of memory used by the I/O buffer cache.

usermem Specifies the amount of free user memory in bytes. This is the amount of memory available after the ULTRIX-11 operating system and the buffer cache.

erase Specifies that you can use the <DELETE> key to delete by character.

kill Specifies that you can press <CTRL/U> to delete an entire line of input.

intr Specifies that you can press <CTRL/C> to interrupt a program.

NOTE

The system prints the last message to remind you that it has changed these control characters from the standard UNIX V7 characters (where erase = #, kill = @, intr = delete).

Once running and ready to accept commands, the ULTRIX-11 operating system issues a # prompt, indicating that the superuser account is active.

3. If you booted an ULTRIX-11 kernel, you need to set the date, check your file systems and the error log file, and then make the system available for time-sharing. For specific information, read Steps 2-4 of Section 3.3, Autobooting from the System Disk.

3.5 Manually Booting from Tape

You normally autoboot the ULTRIX-11 kernel from the root file system on the system disk, unit 0 (for RC25 disks, unit 1). In addition, you can boot from alternate media. Specifically, you can load the boot programs from tape and then manually boot an ULTRIX-11 kernel or a stand-alone program. This procedure lets you boot from a different media. For instructions for booting stand-alone programs, read Appendixes B, D, and F in the ULTRIX-11 Installation Guide.

To load the boot programs from tape and manually boot the ULTRIX-11 kernel, use the following 4-step procedure:

1. Mount the ULTRIX-11 distribution tape on drive 0. Ensure that the tape drive is on-line and ready and that the tape is rewound to load point (BOT).

2. Next, to boot from tape, use either your hardware bootstrap ROM or one of the tape boot programs explained in Appendix A of the ULTRIX-11 Installation Guide.

   Once loaded and running, the secondary boot program prints the following messages:

   ```
   Sizing Memory...

   To list options, type help then press <RETURN>

   Boot:
   ```

3. Enter one of the appropriate boot file specifications. For example, the following command sequence boots the ULTRIX-11 kernel (/unix) from an RM02, unit 0, on the third RH11/RH70 controller:

   ```
   Boot: hj(0,0)unix
   ```

   As it loads the system, the secondary boot program sizes memory and prints this message:

   hj(0,0)unix: size1+size2+size3...sizen

   size1 — Specifies the size of either the ROOT TEXT segment (nonsplit I and D processor) or the DATA+BSS segment (split I and D processor).

   size2 — Specifies the size of either the DATA+BSS segment (nonsplit I and D processor) or the ROOT TEXT segment (split I and D processor).

size3 Specifies the size of the first overlay.

sizen Specifies the size of the last overlay. A maximum of seven overlays may be used and their sizes reported.

You also can boot other files from other disks by entering the appropriate file specification. For further information, read Section 3.2, Specifying the Boot File.

NOTE

If the operating system that you wish to boot is not named /unix, DIGITAL recommends that you rename it /unix. Several user-level commands and system-level programs use the /unix file for the name list of the operating system that currently is running.

If you manually booted the ULTRIX-11 operating system, it starts running in single-user mode and prints this start-up banner:

```
ULTRIX-11 Kernel V#

realmem = ######
buffers = ######
usermem = ######
erase = delete, kill = ^U, intr = ^C
#
```

V# Specifies the version of the ULTRIX-11 system that was booted.

realmem Specifies the size of all of memory in bytes.

buffers Specifies the amount of memory used by the I/O buffer cache.

usermem Specifies the amount of free user memory in bytes. This is the amount of memory available after the ULTRIX-11 operating system and the buffer cache.

erase Specifies that you can use the <DELETE> key to delete by character.

kill Specifies that you can press <CTRL/U> to

delete an entire line of input.

intr Specifies that you can press <CTRL/C> to interrupt a program.

NOTE

The system prints the last message to remind you that it has changed these control characters from the standard UNIX V7 characters (where erase = #, kill = @, intr = delete).

Once running and ready to accept commands, the ULTRIX-11 operating system issues a # prompt, indicating that the superuser account is active.

4. If you booted an ULTRIX-11 kernel, you need to set the date, check your file systems and the error log file, and then make the system available for time-sharing. For specific information, read Steps 2-4 of Section 3.3, Autobooting from the System Disk.

3.6 Manually Booting from RX50 Diskette

On the Micro/PDP-11, you also can boot from different media. Specifically, you can load the boot programs from the boot diskette and then manually boot an ULTRIX-11 kernel or a stand-alone program. For further information, read Appendixes B, D, and F in the ULTRIX-11 Installation Guide.

To load the boot programs from the boot diskette and manually boot the ULTRIX-11 kernel, use the following 4-step procedure:

1. First, insert the boot diskette into the RX50 drive, unit 2.

2. Next, to boot RX50, unit 2, use your hardware bootstrap ROM.

   Once loaded and running, the secondary boot program prints the following messages:

   ```
   Sizing Memory...

   To list options, type help then press <RETURN>

   Boot:
   ```

3. Enter one of the appropriate boot file specifications. For example, the following command sequence boots the ULTRIX-11 kernel (/unix) from an RD51, unit 0:

   ```
   Boot: rd(0,0)unix
   ```

   As it loads the system, the secondary boot program sizes memory and prints this message:

   rd(0,0)unix: size1+size2+size3...sizen

   size1 — Specifies the size of either the ROOT TEXT segment (nonsplit I and D processor) or the DATA+BSS segment (split I and D processor).

   size2 — Specifies the size of either the DATA+BSS segment (nonsplit I and D processor) or the ROOT TEXT segment (split I and D processor).

   size3 — Specifies the size of the first overlay.

   sizen — Specifies the size of the last overlay. A maximum of seven overlays may be used and their sizes reported.

You also can boot other files from other disks by entering the appropriate file specification. For further information, read Section 3.2, Specifying the Boot File.

NOTE

If the operating system that you wish to boot is not named /unix, DIGITAL recommends that you rename it /unix. Several user-level commands and system-level programs use the /unix file for the name list of the operating system that currently is running.

If you manually booted the ULTRIX-11 operating system, it starts running in single-user mode and prints this start-up banner:

```
ULTRIX-11 Kernel V#

realmem = ######
buffers = ######
usermem = ######
erase = delete, kill = ^U, intr = ^C
#
```

V# Specifies the version of the ULTRIX-11 system that was booted.

realmem Specifies the size of all of memory in bytes.

buffers Specifies the amount of memory used by the I/O buffer cache.

usermem Specifies the amount of free user memory in bytes. This is the amount of memory available after the ULTRIX-11 operating system and the buffer cache.

erase Specifies that you can use the <DELETE> key to delete by character.

kill Specifies that you can press <CTRL/U> to delete an entire line of input.

intr Specifies that you can press <CTRL/C> to interrupt a program.

NOTE

The system prints the last message to remind you that it has changed these control characters from the standard UNIX V7 characters (where erase = #, kill = @, intr = delete).

Once running and ready to accept commands, the ULTRIX-11 operating system issues a # prompt, indicating that the superuser account is active.

4. If you booted an ULTRIX-11 kernel, you need to set the date, check your file systems and the error log file, and then make the system available for time-sharing. For specific information, read Steps 2-4 of Section 3.3, Autobooting from the System Disk.

3.7 Boot Program Options

If you abort the autoboot procedure or use a manual boot procedure to load from alternate media, the secondary boot program prints this message and a boot prompt:

```
To list options, type help then press <RETURN>

Boot:
```

Once you receive this prompt, you can obtain information about these boot program options:

- Automatic unit select option
- Automatic CSR select option
- CSR address option

To obtain on-line help, type:

```
Boot: help
```

The secondary boot program then prints the following information and a boot prompt:

```
ULTRIX-11 boot program options.

Enter the desired option using lowercase characters,
then press <RETURN>.  Refer to the ULTRIX-11 System
Management Guide for more detailed information about
each option.  The section number is listed after the
option name.

Option  Section  Description
------  -------  -----------
help    3.7      Print this list of boot options.
aus     3.7.1    Enable/disable auto unit select.
acs     3.7.2    Enable/disable auto CSR select.
csr     3.7.3    List/change device CSR address.

Auto unit select changes the unit number in the ROOT,
PIPE, SWAP, and ERROR LOG specifications in the booted
kernel. This allows booting from disks other than unit 0.

Auto CSR select changes the CSR address for the system
disk to match the CSR in the BOOT program.

The csr command lists and/or changes a device's CSR
address in the BOOT program.
```

The next three sections provide detailed discussions of these options.

3.7.1 Automatic Unit Select Option

During a boot, the secondary boot program passes the unit number of the system disk to the booted ULTRIX-11 kernel and appropriately changes its ROOT, PIPE, SWAP, and ERROR LOG specifications. The automatic unit select option (aus) lets you disable this feature and leave these specifications as originally configured.

To use the aus option, type:

```
Boot: aus
```

The secondary boot program then prints:

```
Auto unit select <y or n> ?
```

To disable automatic unit select, type n. To enable automatic unit select (default), type y.

The aus option lets you boot from a unit other than unit 0 and, for RC25 disks, from a unit other than unit 1.

3.7.2 Automatic CSR Select Option

During a boot, the secondary boot program passes the CSR address listed in its internal table for the system disk to the booted kernel. When booting a stand-alone program, the secondary boot program passes all CSR addresses listed in its internal table to the booted program. The automatic CSR select option (acs) lets you disable this feature and leave each CSR address as originally configured.

To use the acs option, type:

```
Boot: acs
```

The secondary boot program then prints:

```
Auto CSR select <y or n> ?
```

To disable automatic CSR select, type n. To enable automatic CSR select (default), type y.

The acs option lets you boot from a disk that is at a non-standard CSR address.

3.7.3 CSR Address Option

During a boot, the secondary boot program passes the CSR address listed in its internal table for the system disk to the booted kernel. When booting a stand-alone program, the secondary boot program passes all CSR addresses listed in its internal table to the booted program. If any device is

at a nonstandard CSR address, you can use the csr option to change its CSR address in the boot program's internal table.

To use the csr option, type:

```
Boot: csr
```

When this option is specified, the secondary boot program first prints:

```
List all current CSR addresses <y or n> ?
```

To list all current CSR address assignments, type y. The secondary boot program then prints a table of all devices and their corresponding CSR addresses.

Regardless of your response to the previous prompt, the secondary boot program then prints:

```
Enter device name or press <RETURN> for no change.

Device <list of ULTRIX-11 device mnemonics>:
```

To change a device's CSR address in the boot table, type the appropriate ULTRIX-11 mnemonic. The secondary boot program then prints:

```
Enter new CSR address or press <RETURN> for no change.

CSR <old address>:
```

When you enter the new CSR address, the secondary boot program prompts you for verification:

```
New (mnemonic) CSR address is nnnnn <y or n> ?
```

When you respond yes, the secondary boot program changes its CSR table accordingly and prints a boot prompt. If you respond no, the secondary boot program disregards the request.

For example, the following uses the csr option to change the current CSR address for an RC25 disk:

```
Boot: csr
List all current CSR addresses <y or n> ? n
Enter device name or press <RETURN> for no change.
Device <hk hp hm hj ra rc rd rx rk rl rp ht tm ts>: rc
Enter new CSR address or press return for no change.
CSR <172150>: 172160
New (rc) CSR address is 172160 <y or n> ? y
```

3.8 Boot Error Messages

Although the primary boot program does not display error messages, the secondary boot program prints four types of error messages:

- File specification errors
- Operating system parameter errors
- Device errors
- Hardware traps

3.8.1 File Specification Errors

The following error messages indicate an invalid file specification:

```
Bad device
Unknown device
Bad unit specifier
Missing offset specification
Bad magtape file name
```

The following error messages indicate that the secondary boot program could not find the file, that you specified an invalid pathname, or that the file system is corrupted:

```
null path
(filename) not found
not a directory
zero length directory
```

The following error messages indicate either that the secondary boot program could not read the file system on the device or that the file system is corrupted:

```
bn negative
bn ovf # (# = block number)
bn void # (# = block number)
```

3.8.2 Operating System Parameter Errors

When it detects a parameter error while loading the operating system file (normally /unix), the secondary boot program prints one of more of the following error messages. These messages usually indicate that the operating system could not be booted. Because the sysgen program checks all of the system parameters during system generation, these errors are not likely to occur.

The following messages indicate either that the processor has insufficient memory or that the system was generated with too few buffers:

```
less than 192K bytes of memory!
less than 16 buffers!
```

The following messages indicate that the system's name list (symbol table) either is missing or could not be accessed:

```
Unix symbol table missing
Can't access namelist in xxxx
```

The following error messages indicate either that the system file could not be opened for reading or that it is not the correct file type. The system must be a 0431 (split I & D) or 0430 (text overlay) file. The # indicates the actual type of the file being loaded:

```
Can't load (#) unix files
Can't load # files
Can't open xxxx file
```

The following message indicates that the secondary boot program attempted to load a split I & D kernel on a nonseparate I & D processor: The # indicates the file type.

```
Can't load sep I&D (#) files
```

The following messages indicate that the size of one of the system's segments is invalid. The # indicates the size of the segment in octal:

```
text segment too small (#)
text segment too big (#)
data segment too large (#)
data segment too big (#)
max overlay too big (#)
Root text segment too small (#)
Root text segment too large (#)
```

The following message indicates a problem with the mapped I/O buffer system in the overlay kernel. For further information, read Section 2.7.4, MAPPED BUFFERS - forbidden zone violation! The # indicates the value of the symbol _mb_end in octal. This value is used to correct the error condition.

```
MAPPED BUFFERS - forbidden zone violation
_mb_end = #
```

The following message indicates a problem with the UNIBUS mapping. For further information, read Section 2.7.5, UNIBUS MAP - forbidden zone violation! The # indicates the value of the symbol _ub_end in octal. This value is used to correct the error condition.

```
UNIBUS MAP - forbidden zone violation!
_ub_end = #
```

The following message indicates that the TTY structures could not be assigned to the communications interface ports because one of the parameters needed to make the assignment either is out of range or could not be found:

```
Can't assign TTY structures in xxxx
```

The following messages indicate that the system has been generated with too many buffers. The # indicates the amount of memory available:

```
TOO MANY BUFFERS for #K bytes of memory!
UNIBUS MAP REGISTER LIMIT EXCEEDED: too many buffers!
```

When these messages are printed, the secondary boot program makes the appropriate adjustments and then prints this message:

```
Reducing number of buffers to #
```

The # indicates the current number of buffers.

3.8.3 Device Errors

All ULTRIX-11 stand-alone device drivers report errors in the following format:

- Header -- device name and unit number
- Address -- block number or disk address
- Hardware registers -- control, status, and error
- Error type -- fatal or recoverable

For UDA50, KLESI, RUX1, and RQDX1 disk errors, an MSCP error code prints instead of the hardware registers. For further information, read Appendix F, UDA50/KLESI/RUX1/RQDX1 - MSCP Error Codes.

For example, the following message indicates that a fatal error occurred on an RK06/7, unit 2:

```
HK unit 2 disk error:
cs1=100220 cs2=202 ds=100301 err=200 hkdc=293 track=1 sect=12
(FATAL ERROR)
```

The stand-alone drivers for disks with hardware Error Correction Capability print a message in the following format when the driver uses ECC to recover data from a disk block:

```
HP # ECC bn = ##
```

The # indicates the unit number and ## indicates the logical block number.

3.8.4 Hardware Traps

When hardware-detected errors occur, the PDP-11 processors trap through fixed addresses in low memory. The possible processor trap locations are:

Location	Error type
4	bus error
10	illegal instruction
14	break point trace
20	IOT instruction
24	power fail
30	EMT instruction
34	TRAP instruction
240	Programmed interrupt request
244	floating point exception
250	memory management segmentation
114	memory parity
???	stray vector

The stand-alone ULTRIX-11 device drivers do not operate in interrupt mode. The stray vector error message indicates the processor vectored through an unexpected location (other than one of the above trap addresses). Any device interrupt causes the stray vector message.

The following indicates that a UNIBUS timeout trap occurred. This probably was caused by attempting to boot from a nonexistent device. The value in R2 (172150) is the device address for the UDA50 disk controller:

```
Boot: ra(0,0)unix

Trap - bus error

ps = 140344
pc = 14672
r0 = 14604
r1 = 0
r2 = 172150
r3 = 52070
r4 = 52070
r5 = 63642
sp = 157750
```

Chapter 4

ULTRIX-11 Maintenance and Administrative Functions

To operate and maintain an ULTRIX-11 system, either the system manager or operator regularly should:

- Check file system consistency
- Correct reported file system inconsistencies
- Back up file systems
- Check for dynamic bad blocks on MSCP disks
- Monitor general system performance

Then, as the need arises, the system manager should:

- Add new user accounts
- Set up additional user file systems
- Enable user terminals
- Set up the cu facility
- Set up tip connections
- Install the uucp facility

The remaining sections of this chapter provide discussions of these tasks.

NOTE

DIGITAL recommends that the system operator use the opser program for normal system maintenance. For further information, read Chapter 5, ULTRIX-11 Operator Services.

4.1 File System Maintenance

As system manager, you should monitor the status of your file systems on a regular basis. Specifically, you regularly should:

- Check file system consistency
- Correct all reported file system inconsistencies
- Report disk free space
- Report disk usage
- Check disk quotas

4.1.1 Checking File System Consistency

As system manager, you should ensure that all file systems are checked regularly. To check the consistency of your file systems, use the fsck command. Before doing so, however, you should unmount all file systems and shut down multiuser mode.

Once the system is in single-user mode, use the fsck command to check the consistency of your file systems. In response to the superuser prompt, type this command sequence:

```
# fsck special
```

special — Specifies the logical special file name of the file system that you want to check. If you do not specify a file system, the fsck command automatically checks all file systems that are listed in /etc/fstab.

When executed on a given file system, the fsck command:

- Checks the file system
- Reports all inconsistencies
- Reports the current number of files
- Reports the number of blocks used
- Reports the current number of free blocks

NOTE

When checking the root file system (/), you should use the block mode special file name. When checking any other file systems, you should use only a character mode special file name.

The following command sequence checks both the root and /usr file system on an RL02 system disk:

```
# fsck /dev/rl00 /dev/rrl01
```

When checking large file systems, the fsck command may prompt you for the name of a scratch file. You should respond with the name of the file that you want to use (for example, /tmp/fsck.scratch) and press the <RETURN> key. When checking multiple file systems, you should specify the scratch file name with the -t option. For example, the following command sequence checks and reports on all file systems listed in /etc/fstab and uses /tmp/fsck.scratch as the scratch file:

```
# fsck -t /tmp/fsck.scratch
```

For further information, read fsck(1M) in the ULTRIX-11 Programmer's Manual, Volume 1.

NOTE

DIGITAL recommends the system operator use the opser program to check file systems. For further information, read Chapter 5, ULTRIX-11 Operator Services.

4.1.2 Correcting File System Inconsistencies

When it detects an inconsistency, the fsck command prints a message that describes the type of inconsistency and a prompt (?) for your response to the suggested course of action.

When you respond yes, the fsck command attempts to correct the inconsistency. When you respond no (or anything else), the fsck command leaves the inconsistency uncorrected. In either case, the fsck command continues to check and report on the designated file systems. When you respond yes to a prompt about an orphaned file (no directory entry or link), the fsck command relinks the file to the lost+found directory in that file system.

The ULTRIX-11 system has several other commands that you can also use to check for and correct file system inconsistencies:

dcheck — Checks the consistency of the directory hierarchy on the specified file system.

fsdb — Provides a means of correcting those file system inconsistencies that fsck cannot.

icheck — When the -b option is specified, reports

information on the named block. This command is useful for gathering information about the inode to which the block is allocated.

ipatch Prints the contents of an inode. This command can also be used to change this information.

ncheck When the -i option is specified, traces the named inode back to a file name.

NOTE

DIGITAL recommends that you use the ipatch command to change the contents of an inode only if you have a thorough understanding of the ULTRIX-11 file system.

For further information, read dcheck(1M), fsck(1M), fsdb(1M), icheck(1M), ipatch(1M), and ncheck(1M) in the ULTRIX-11 Programmer's Manual, Volume 1.

4.1.3 Reporting Disk Free Space

To report the current number of free blocks available on a file system, type this command sequence:

```
$ df special
```

special Specifies the logical special file name that contains the file system. You can type more than one name. If you do not specify a special file name, the df command reports the free space on all mounted file systems listed in /etc/fstab.

The following command sequence reports the current amount of free space on both file systems on an RL02 disk, unit 0:

```
$ df /dev/rrl00 /dev/rrl01
```

For further information, read df(1M) in the ULTRIX-11 Programmer's Manual, Volume 1.

4.1.4 Reporting Disk Usage

To report user disk (directory) usage, type this command sequence:

```
$ du directory
```

directory — Specifies the directory for which the disk usage summary is to be generated. You can type more than one name. If you do not specify a directory name, the du command reports the disk usage for the current directory.

The following command sequence reports the disk usage for all main subdirectories in the /usr/staff file system:

```
$ du /usr/staff
```

For further information, read du(1) in the ULTRIX-11 Programmer's Manual, Volume 1.

4.1.5 Checking Disk Quotas

To report the number of blocks owned by each user of a file system, type this command sequence:

```
$ quot special
```

special — Specifies the logical special file name that contains the file system.

The following command sequence reports the number of blocks owned by each user on the /usr file system:

```
$ quot /dev/rrl01
```

For further information, read quot(1M) in the ULTRIX-11 Programmer's Manual, Volume 1.

4.2 Backing Up and Restoring File Systems

DIGITAL recommends that you use the opser program for normal file system backups and for shutting down multiuser mode prior to restoring file systems. For further information, read Chapter 5, ULTRIX-11 Operator Services.

The ULTRIX-11 system is distributed with prototype shell command files that are to be used with the opser backup command. These shell command files are:

daily.bak	Contains the prompts and commands that are to be used to perform daily file system backups on ULTRIX-11 PDP/11 systems.
monthly.bak	Contains the prompts and commands that are to be used to perform monthly file system backups on ULTRIX-11 PDP/11 systems.
m11_day.bak	Contains the prompts and commands that are to be used to perform daily file system backups on ULTRIX-11 Micro/PDP-11 systems.
m11_week.bak	Contains the prompts and commands that are to be used to perform weekly file system backups on ULTRIX-11 Micro/PDP-11 systems.

As system manager, you must tailor the appropriate command files to your system configuration. To tailor the files that are to used for your system, use one of the ULTRIX-11 editors and make the necessary changes. Specifically, verify or, as needed, change and add the device special file names for your file systems. Once tailored to your site, instruct the system operator of the schedule and file names that are to be used with the opser backup command.

As distributed, these shell command files use the dump command to perform incremental backups on a monthly, weekly, or daily basis. Once a month or once a week on the Micro/PDP-11, a shell file backs up all files on all file systems. Once a day, a shell file only backs up those files that have been modified from the time of the monthly or weekly backup. The number of files that are backed up daily depends both on the size of your user community and on the amount of file system activity.

NOTE

When the number of files that are backed up daily becomes unmanageable, DIGITAL recommends using either the monthly or weekly command file ahead of schedule. These command files back up all files and, therefore, reduce the number of files that are dumped on subsequent daily backups.

To restore individual files or entire file systems that have been dumped to tape, you must use the restor command. When restoring individual files, you may use the restor commands while the system is in multiuser mode. When restoring entire file systems, however, DIGITAL recommends first using the opser command to shut down multiuser mode and then using the restor command to restore the file system.

The ULTRIX-11 system has several other commands that you can use for special purpose backups or restores:

cp — Copies user-specified files to or from a backup file. This command is useful for copying a minimal number of files to another directory or file system.

cpio — Copies user-specified files or entire file systems to or from backup media. This command is useful for creating file archives.

tar — Copies user-specified files to or from a backup tape. This command is useful for creating a tape archive or for copying a minimal number of files to tape.

volcopy — Copies an entire disk image to or from a backup disk. This command is useful for making an exact image copy of a file system.

NOTE

When using RX50 diskettes, the number and size of the files copied is limited to the size of the diskette (800 blocks minus file system overhead). Because both commands support multivolumes, dump and restor overcome this size limitation. When using a diskette with either the dump or restor commands, you should specify the m key. This key specifies that RX50 diskettes are being used in place of tape.

For further information, read cp(1), cpio(1), dump(1M), dumpdir(1M), restor(1M), tar(1), and volcopy(1M) in the ULTRIX-11 Programmer's Manual, Volume 1.

4.2.1 Restoring the Root File System

To restore an individual file to the root file system, you should use the restor command. To restore the entire root file system, however, you should use this procedure:

1. Use the stand-alone mkfs program and make a new file system. This program creates an empty file system on the root partition of the system disk. It is important to remember that the mkfs program overwrites any existing data as it makes the new file system.

2. Use the stand-alone restor program to restore the root file system from backup media. Restore the full (level 0) dump onto the root partition of the system disk.

3. If there is a later level 9 (incremental) dump of the root file system, use the stand-alone restor program to restore those files.

4. Use the stand-alone icheck program to check the restored root file system for inconsistencies.

5. Boot the restored root file system.

For further information, read Appendix D, Disk Logical Partition Sizes, and Appendix F, Stand-alone Programs, in the ULTRIX-11 Installation Guide.

4.2.2 Backing Up File Systems to TK25 Tape

On a Micro/PDP-11 system, you normally use the opser backup command with the appropriate command file specified to dump your file systems to an RX50 diskette. On a Micro/PDP-11 system, however, you also may dump file systems to TK25 tape units. The ULTRIX-11 system, therefore, is distributed with a prototype shell command file that may be used with the opser command for this purpose:

tk_daily.bak Contains the prompts and commands that are to be used to perform daily file system backups to TK25 tape.

When using TK25 tape in place of RX50 diskettes, you must consider:

- The TK25 unit first moves through to the end and rewinds the cartridge every time that it is loaded to ensure that the cartridge is at the proper tension. This may take up to five minutes.

- This shell command file dumps multiple file systems on one TK25 cartridge to take advantage of its large capacity. Specifically, it dumps each file system as a separate tape file. To do so, it dumps each file system (except the last) to the specified special file with no rewind enabled. For further information, read Section 1.4.4, Magnetic Tapes.

NOTE

To create a TK25 shell command file that may be used to perform weekly backups, copy the tk_daily.bak file. Then, use one of the ULTRIX-11 editors to change the dump levels to level 0. Finally, use the mv command to rename the file tk_weekly.bak.

4.2.3 Restoring Files from TK25 Backups

To restore a file system from TK25 tape, you first must position the cartridge at the beginning of the appropriate tape file. To do so, use the dd command and specify the following keywords:

if= Indicate the TK25 special file name with no rewind enabled (for example, /dev/nrht0).

of= Indicate /dev/null. The input is discarded.

bs= Indicate 20b for an input and output blocking factor of 20 blocks.

files= Indicate the number of tape files that are to be skipped.

For example, the following command sequence positions the cartridge at the beginning of the second tape file:

```
dd if=/dev/nrht0 of=/dev/null bs=20b files=1
```

To position the cartridge at the beginning of the third tape file, substitute files=2 in the previous command sequence.

NOTE

When using the dumpdir command to list the contents of a tape file, the TK25 unit automatically rewinds the cartridge. Before using the restor command, therefore, you must reposition the cartridge at that tape file.

For further information, read dd(1) in the ULTRIX-11 Programmer's Manual, Volume 1.

4.2.4 Preparing for RC25 File System Backups

If you are using magnetic tape or TK25 cartridge as your backup media, use the appropriate opser backup file to back up your RC25 file systems. If you are using RC25 cartridges as your backup media, use the backup procedure explained in Section 4.2.5, Backing Up RC25 File Systems.

Before you can back up your RC25 file systems, however, you must initialize the backup disk cartridges. If you initialize a sufficient number of cartridges once, you do not have to repeat this step prior to every backup. DIGITAL recommends that you initialize a minimum of four backup RC25 cartridges and label them:

System disk backup A
User disk backup A
System disk backup B
User disk backup B

If you initialize at least four disk cartridges, you can rotate each set daily.

To initialize the backup cartridges, use this procedure:

1. Shut down multiuser mode by using the opser program and

specifying the s command. Once the system is in single-user mode, check all file systems on the system and user disks by specifying the f command. Then, halt the processor by specifying the opser halt command. For further information, read Chapter 5, ULTRIX-11 Operator Services.

2. Execute the system's hardware boot ROM and wait for the message:

    (<CTRL/C> aborts the auto-boot)

    Quickly, abort the autoboot procedure by pressing <CTRL/C>. Once you receive a boot prompt, type:

    Boot: rc(1,0)/sas/rabads

    This command boots the stand-alone rabads program from the system disk (unit 1).

3. Once running, the rabads program prints this program banner and command prompt:

    ULTRIX-11 MSCP Disk Initialization Program

    rabads <help exit drives status table init replace>:

    Once you receive this command prompt, initialize each backup cartridges by specifying the init command. When you specify the init command, the rabads program issues the following prompts to which you respond:

    disk type <ra60 ra80 ra81 rx50 rd51 rd52 rc25>: rc25

    unit number <0-7>: 0

    Starting block number <0>: 0

    Number of blocks to check <50902>: 50902

    Rewrite blocks written with "forced error" <y or n> ? y

    As it initializes each cartridge, the rabads program prints the number of blocks checked, bad blocks found, and bad blocks replaced. After you have initialized all cartridges, exit the rabads program by specifying the exit command. For further information, read Appendix B, Disk Media Qualification, in the ULTRIX-11 Installation Guide.

4. If you want to back up your RC25 file systems immediately, proceed to Step 2 in Section 4.2.5, Backing Up RC25 File Systems, and boot the stand-alone copy program as explained. If you do not want to back up your RC25

file systems immediately, reboot your ULTRIX-11 system by executing the system's hardware boot ROM. For further information, read Section 3.3, Autobooting from the System Disk.

Once you have initialized a sufficient number of RC25 cartridges, you are ready to back up your RC25 file systems.

4.2.5 Backing Up RC25 File Systems

To back up your RC25 file systems, you use the stand-alone copy program to create an image copy of your system and user disks. DIGITAL recommends that you back up your RC25 file systems daily and that you also alternate the backup cartridges (sets A and B). Depending on your processor type, this procedure may take up to 45 minutes to complete.

Throughout this procedure, you are repeatedly told to load and remove RC25 cartridges. When told to load a cartridge, you should:

1. Insert the cartridge in the drive
2. Close the door
3. Press the RUN switch
4. Wait for the RUN light to stop flashing

When you are told to remove a cartridge, you should:

1. Release the RUN switch
2. Wait for the RUN light to stop flashing
3. Press the eject switch
4. Remove the cartridge

To back up your RC25 file systems, use this procedure:

1. Shut down multiuser mode by using the opser program and specifying the s command. Once the system is in single-user mode, check all file systems on the system and user disks by specifying the f command. Then, halt the processor by specifying the opser halt command. For further information, read Chapter 5, ULTRIX-11 Operator Services.

2. Execute the system's hardware boot ROM and wait for the message:

   (<CTRL/C> aborts the auto-boot)

   Quickly, abort the autoboot procedure by pressing <CTRL/C>. Once you receive a boot prompt, type:

   Boot: rc(1,0)/sas/copy

   This command sequence boots the stand-alone copy

program from the system disk (unit 1). Once running, the stand-alone copy program prints the following message and prompts for all required information:

```
ULTRIX-11 Standalone Copy Program
```

It first prompts for the input and output file specifications, the record size, and the number of records. Then, it prompts for verification before beginning to copy the specified file.

3. Prepare to copy the system disk to a backup cartridge by removing the user disk from unit 0, loading the backup system disk cartridge into unit 0, write-protecting unit 1, and write-enabling unit 0.

4. Copy the system disk (unit 1) to the backup system disk cartridge (unit 0) by answering each prompt:

```
Input File: rc(1,0)

Output File: rc(0,0)

Record Size <16384 MAX>: 10240

Number of Records: 2540

Ready to copy from rc(1,0) to rc(0,0) <y or n> ? y
```

The stand-alone copy program takes about five minutes to copy the system disk to the backup cartridge. When it is done, the stand-alone copy program prints a completion message and a continuation prompt:

```
Copy complete

More files to copy <y or n> ?
```

Answer yes.

5. Prepare for the next copy by removing the backup system disk cartridge from unit 0, loading the user disk into unit 0, write-protecting unit 0, and write-enabling unit 1.

6. Copy the user disk (unit 0) to unit 1 by answering each prompt:

```
Input File: rc(0,0)

Output File: rc(1,0)

Record Size <16384 MAX>: 10240
```

```
Number of Records: 2540

Ready to copy from rc(0,0) to rc(1,0) <y or n> ? y
```

The stand-alone copy program takes about five minutes to copy the the user disk to unit 1. When it is done, the stand-alone copy program prints a completion message and a continuation prompt:

```
Copy complete

More files to copy <y or n> ?
```

Answer yes.

7. Prepare for the next copy by removing the user disk from unit 0, loading the backup user disk cartridge into unit 0, write-protecting unit 1, and write-enabling unit 0.

8. Copy unit 1 to the backup user disk cartridge (unit 0) by answering each prompt:

```
Input File: rc(1,0)

Output File: rc(0,0)

Record Size <16384 MAX>: 10240

Number of Records: 2540

Ready to copy from rc(1,0) to rc(0,0) <y or n> ? y
```

The stand-alone copy program takes about five minutes to copy unit 1 to the backup user disk cartridge. When it is done, the stand-alone copy program prints a completion message and a continuation prompt:

```
Copy complete

More files to copy <y or n> ?
```

Answer yes.

9. Prepare for the next copy by removing the backup user disk cartridge from unit 0, loading the backup system disk cartridge into unit 0, write-protecting unit 0, and write-enabling unit 1.

10. Copy the backup system disk cartridge (unit 0) to unit 1 by answering each prompt:

```
Input File: rc(0,0)
```

```
Output File: rc(1,0)

Record Size <16384 MAX>: 10240

Number of Records: 2540

Ready to copy from rc(0,0) to rc(1,0) <y or n> ? y
```

The stand-alone copy program takes about five minutes to copy the backup system disk to unit 1. When it is done, the stand-alone copy program prints a completion message and a continuation prompt:

```
Copy complete

More files to copy <y or n> ?
```

Answer no.

11. Having recopied the system disk prepare to reboot by removing the backup system disk cartridge from unit 0, loading the user disk into unit 0, write-enabling unit 1 and unit 0.

12. Reboot your ULTRIX-11 system by executing your system's hardware boot ROM. For further information, read Section 3.3, Autobooting from the System Disk.

4.2.6 Restoring Individual Files from RC25 Backups

To restore individual files from an RC25 system or user disk backup, you use the cp command to copy the files from the RC25 backup to a temporary directory.

To restore individual files from an RC25 backup, use this procedure:

1. Shut down multiuser mode by using the opser program and specifying the s command. Once the system is in single-user mode, check all file systems on the system and user disks by specifying the f command. Escape the opser program by specifying the !sh command. For further information, read Chapter 5, ULTRIX-11 Operator Services.

2. Prepare to restore the files by removing the user disk from unit 0, loading the backup disk cartridge into unit 0, write-protecting unit 0, and write-enabling unit 1.

3. Because an RC25 backup is a disk image copy, you must mount each file system containing files that are to be restored, one at a time, onto the /mnt directory. You do so by using the mount command with the -r option specified. Once a file system is accessible, copy the

files by using the cp command. After the files are copied from each file system, unmount that file system by using the umount command. Repeat this step until you have copied all the files that are to be restored. For further information, read cp(1) and mount(1M) in the ULTRIX-11 Programmer's Manual, Volume 1.

4. Prepare to restart multiuser mode by removing the backup cartridge from unit 0, loading the user disk into unit 0, and write-enabling unit 0 and unit 1.

5. Return to the opser program by pressing <CTRL/D>. Once you receive an opr> prompt, restart multiuser mode by specifying the r command.

6. Once the system is in multiuser mode, copy the restored files to the directories that they belong in by using the cp command.

4.2.7 Restoring the System Disk from RC25 Backups

To restore the system disk from an RC25 system disk backup, you use the stand-alone copy program to copy the backup image to the system disk. Throughout this procedure, you are repeatedly told to load and remove RC25 cartridges.

When told to load a cartridge, you should:

1. Insert the cartridge in the drive
2. Close the door
3. Press the RUN switch
4. Wait for the RUN light to stop flashing

When you are told to remove a cartridge, you should:

1. Release the RUN switch
2. Wait for the RUN light to stop flashing
3. Press the eject switch
4. Remove the cartridge

To restore your system disk, use this procedure:

1. Shut down multiuser mode by using the opser program and specifying the s command. Once the system is in single-user mode, halt the processor by specifying the opser halt command. For further information, read Chapter 5, ULTRIX-11 Operator Services.

2. Prepare to restore the system disk by removing the user disk from unit 0, loading the backup system disk cartridge into unit 0, write-protecting unit 0, and write-enabling unit 1.

3. Execute the system's hardware boot ROM and wait for the

message:

```
(<CTRL/C> aborts the auto-boot)
```

Quickly, abort the autoboot procedure by pressing <CTRL/C>. Once you receive a boot prompt, type:

```
Boot: rc(0,0)/sas/copy
```

This command sequence boots the stand-alone copy program from the system disk (unit 0). Once running, the stand-alone copy program prints the following message and prompts for all required information:

```
ULTRIX-11 Standalone Copy Program
```

It first prompts for the input and output file specifications, the record size, and the number of records. Then, it prompts for verification before beginning to copy the specified file.

4. Copy the backup system disk cartridge (unit 0) to unit 1 by answering each prompt:

```
Input File: rc(0,0)

Output File: rc(1,0)

Record Size <16384 MAX>: 10240

Number of Records: 2540

Ready to copy from rc(0,0) to rc(1,0) <y or n> ? y
```

The stand-alone copy program takes about five minutes to copy the backup system disk to unit 1. When it is done, the stand-alone copy program prints a completion message and a continuation prompt:

```
Copy complete

More files to copy <y or n> ?
```

Answer no.

5. Having restored the system disk prepare to reboot by removing the backup system disk cartridge from unit 0, loading the user disk into unit 0, write-enabling unit 1 and unit 0.

6. Reboot your ULTRIX-11 system by executing your system's hardware boot ROM. For further information, read Section 3.3, Autobooting from the System Disk.

4.2.8 Restoring the User Disk from RC25 Backups

To restore the user disk from an RC25 user disk backup, you use the stand-alone copy program. Throughout this procedure, you are repeatedly told to load and remove RC25 cartridges.

When told to load a cartridge, you should:

1. Insert the cartridge in the drive
2. Close the door
3. Press the RUN switch
4. Wait for the RUN light to stop flashing

When you are told to remove a cartridge, you should:

1. Release the RUN switch
2. Wait for the RUN light to stop flashing
3. Press the eject switch
4. Remove the cartridge

To restore the user disk from an RC25 backup, use this procedure:

1. Shut down multiuser mode by using the opser program and specifying the s command. Once the system is in single-user mode, check all file systems on the system and user disks by specifying the f command. Then, halt the processor by specifying the opser halt command. For further information, read Chapter 5, ULTRIX-11 Operator Services.

2. Execute the system's hardware boot ROM and wait for the message:

   (<CTRL/C> aborts the auto-boot)

   Quickly, abort the autoboot procedure by pressing <CTRL/C>. Once you receive a boot prompt, type:

   Boot: rc(1,0)/sas/copy

   This command sequence boots the stand-alone copy program from the system disk (unit 1). Once running, the stand-alone copy program prints the following message and prompts for all required information:

   ULTRIX-11 Standalone Copy Program

   It first prompts for the input and output file specifications, the record size, and the number of records. Then, it prompts for verification before beginning to copy the specified file.

3. Prepare to copy the system disk to a backup cartridge by removing the user disk from unit 0, loading the backup system disk cartridge into unit 0, write-protecting unit 1, and write-enabling unit 0.

4. Copy the system disk (unit 1) to the backup system disk cartridge (unit 0) by answering each prompt:

```
Input File: rc(1,0)

Output File: rc(0,0)

Record Size <16384 MAX>: 10240

Number of Records: 2540

Ready to copy from rc(1,0) to rc(0,0) <y or n> ? y
```

   The stand-alone copy program takes about five minutes to copy the system disk to the backup cartridge. When it is done, the stand-alone copy program prints a completion message and a continuation prompt:

```
Copy complete

More files to copy <y or n> ?
```

   Answer yes.

5. Prepare for the next copy by removing the backup system disk cartridge from unit 0, loading the backup user disk into unit 0, write-protecting unit 0, and write-enabling unit 1.

6. Copy the backup user disk (unit 0) to unit 1 by answering each prompt:

```
Input File: rc(0,0)

Output File: rc(1,0)

Record Size <16384 MAX>: 10240

Number of Records: 2540

Ready to copy from rc(0,0) to rc(1,0) <y or n> ? y
```

   The stand-alone copy program takes about five minutes to copy the the backup user disk to unit 1. When it is done, the stand-alone copy program prints a completion message and a continuation prompt:

```
Copy complete
```

```
More files to copy <y or n> ?
```

Answer yes.

7. Prepare for the next copy by removing the backup user disk cartridge from unit 0, loading the user disk cartridge into unit 0, write-protecting unit 1, and write-enabling unit 0.

8. Copy unit 1 to the user disk cartridge (unit 0) by answering each prompt:

```
Input File: rc(1,0)

Output File: rc(0,0)

Record Size <16384 MAX>: 10240

Number of Records: 2540

Ready to copy from rc(1,0) to rc(0,0) <y or n> ? y
```

The stand-alone copy program takes about five minutes to copy unit 1 to the user disk cartridge. When it is done, the stand-alone copy program prints a completion message and a continuation prompt:

```
Copy complete

More files to copy <y or n> ?
```

Answer yes.

9. Prepare for the next copy by removing the user disk cartridge from unit 0, loading the backup system disk cartridge into unit 0, write-protecting unit 0, and write-enabling unit 1.

10. Copy the backup system disk cartridge (unit 0) to unit 1 by answering each prompt:

```
Input File: rc(0,0)

Output File: rc(1,0)

Record Size <16384 MAX>: 10240

Number of Records: 2540

Ready to copy from rc(0,0) to rc(1,0) <y or n> ? y
```

The stand-alone copy program takes about five minutes to copy the backup system disk to unit 1. When it is done, the stand-alone copy program prints a completion

message and a continuation prompt:

```
Copy complete

More files to copy <y or n> ?
```

Answer no.

11. Having recopied the system disk prepare to reboot by removing the backup system disk cartridge from unit 0, loading the user disk into unit 0, write-enabling unit 1 and unit 0.

12. Reboot your ULTRIX-11 system by executing your system's hardware boot ROM. For further information, read Section 3.3, Autobooting from the System Disk.

4.3 Dynamic Bad Blocks on MSCP Disks

During an ULTRIX-11 installation, you qualify your disk media to ensure that all bad blocks are replaced before each disk unit is used by the ULTRIX-11 system. In addition, during normal operations, the MSCP disks (RX50, RD51, RD52, RC25, RA60, RA80, RA81) report dynamic bad blocks.

After initialization, the MSCP disks report new bad blocks by entering an error record in the system error log file. These error records can be identified by the following flags:

Bad Block Reported	Indicates that the reported logical block either has gone bad or is in the process of going bad. This flag occurs when a read operation on that block results either in an unrecoverable error or in a recoverable error that warrants attention. The reported block number is relative to the start of the physical disk unit.
Bad Block Unreported	Indicates that more than one bad block was encountered. This flag occurs when a read operation was performed on multiple blocks. The reported block number is relative to the start of the physical disk unit and is that of the first bad block. A second read, beginning at the block after the reported block, is required to find the unreported block numbers.
Force Error Modifier	Indicates that, during a replacement operation, the data from the reported block could not be successfully transferred. During a bad block replacement operation, the data is first read from the bad block and then written to the replacement block. This flag occurs when valid data could not be read from the bad block and warns that the data written to the replacement block may be corrupted.

To check for dynamic bad blocks, use the elp command and specify the -s option to print a summary of the error log file. For further information, read Section 8.3.1, Summary and Full Error Reports.

When this report is printed, look for errors on MSCP disks. If there are any, print a full report for the errors on each MSCP disk. Specifically, use the elp program and specify

the -d and -et options to limit the report to the appropriate time range and MSCP disk. For further information, read Section 8.3.3, Error Reports by Error Type, and Section 8.3.5, Error Reports by Date and Time.

For example, this command sequence prints a full report for the any errors that occurred on RA60/RA80/RA81 disks between 12:00 am, January 1, 1984 and 11:59 pm, January 2, 1984:

```
elp -ra -d 840101000000 840102115959
```

NOTE

You can check the error log file for occurrences of dynamic bad blocks on MSCP disks at any time. DIGITAL recommends that you or the system operator check the error log file each time that multiuser mode is shut down to back up your file systems.

Although the exact procedure that you use to correct bad blocks is disk dependent, most involve using the stand-alone rabads program. You can use the rabads program to replace one block, to scan the disk and replace all bad blocks, or to rewrite a replacement block. The rabads program provides on-line help information. For further information, read Appendix G, Rabads Program Example.

NOTE

After the ULTRIX-11 system has been installed, you can use the rabads init command to replace bad blocks on your system disk. When scanning either the entire disk or a specified section, the rabads program reads the existing data from the disk. It does not write a test pattern.

4.3.1 Correcting Bad Blocks on RX50 Disks

Because the RX50 has no means of replacing bad blocks, the diskette must be replaced if a dynamic bad block is reported. If the diskette contains data, therefore, you must copy the data on to another diskette.

Depending on the amount of data, you either can make an image copy of the diskette or can copy individual files from

it. If the image copy fails, try it again. If it still fails after several tries, then you must copy each file individually. By doing so, you will minimize the amount of data that will be lost.

4.3.2 Correcting Bad Blocks on RD51/RD52 Disks

The RQDX1 disk controller provides controller initiated bad block replacement. It automatically detects and replaces dynamic bad blocks. When it does so, the RQDX1 controller logs an error record containing the "Bad Block Reported" flag. If the read of the bad block could not be successfully completed, the controller sets the "Force Error Modifier" in the replacement block. This flag warns that the data written to the replacement block may be corrupted.

To clear the "Force Error Modifier," you must rewrite the replacement block. To do so, use this procedure:

1. Shut down multiuser mode by using the opser program and specifying the s command. Once the system is in single-user mode, check all file systems on the system and user disks by specifying the f command. Escape the opser program by specifying the !sh command. For further information, read Chapter 5, ULTRIX-11 Operator Services.

2. Once you receive a shell prompt, determine the bad block's inode number by using the icheck command with the -b option specified. Then, determine the corresponding file name by using the ncheck command with the -i option specified. For further information, read icheck(1M) and ncheck(1M) in the ULTRIX-11 Programmer's Manual, Volume 1.

3. Mount the file system that contains the file by using the mount command. Once the file system is mounted, determine the amount of corruption by attempting to read through the file. If possible, correct corrupted data by restoring the file from backup media. For further information, read mount(1M) and restor(1M) in the ULTRIX-11 Programmer's Manual, Volume 1.

4. Once the data has been sufficiently restored, quit the shell and return to the opser program by pressing <CTRL/D>. Once receiving an opr> prompt, halt the processor the processor by specifying the halt command.

5. Execute the system's hardware boot ROM and wait for the message:

   (<CTRL/C> aborts the auto-boot)

   Quickly, abort the autoboot procedure by pressing

<CTRL/C>. Once you receive a boot prompt, type:

```
Boot: rd(0,0)/sas/rabads
```

This command boots the stand-alone rabads program from the system disk (unit 0). Once running, the rabads program prints this program banner and command prompt:

```
ULTRIX-11 MSCP Disk Initialization Program

rabads <help exit drives status table init replace>:
```

Once you receive this command prompt, force the replacement block to be rewritten by specifying the replace command. This clears the "Force Error Modifier" in the replacement block. The rabads program provides on-line help information. Once the bad blocks have been replaced, exit the rabads program by specifying the exit command. For further information, read Appendix G, Rabads Program Example.

6. Reboot your ULTRIX-11 system by executing your system's hardware boot ROM. For further information, read Section 3.3, Autobooting from the System Disk.

4.3.3 Correcting Bads Blocks on RC25/RA60/RA80/RA81 Disks

When the controllers for RC25, RA60, RA80, and RA81 disks detect dynamic bad blocks, they log an error record with the "Bad Block Reported" flag set. By checking the system error log file regularly, you then can determine not only when bad blocks occur but also the bad block number. On learning of a dynamic bad block on these MSCP disks, you must replace it.

To replace a dynamic bad block on a RC25, RA60, RA80, and RA81 disk, use this procedure:

1. Shut down multiuser mode by using the opser program and specifying the s command. Once the system is in single-user mode, halt the processor by specifying the halt command. For further information, read Chapter 5, ULTRIX-11 Operator Services.

2. Execute the system's hardware boot ROM and wait for the message:

```
(<CTRL/C> aborts the auto-boot)
```

Quickly, abort the autoboot procedure by pressing <CTRL/C>. Once you receive a boot prompt, type one of these commands:

```
Boot: rc(1,0)/sas/rabads          (RC25)
```

Boot: **ra(0,0)/sas/rabads** (RA60/RA80/RA81)

This command boots the stand-alone rabads program from the system disk. Once running, the rabads program prints this program banner and command prompt:

```
ULTRIX-11 MSCP Disk Initialization Program

rabads <help exit drives status table init replace>:
```

Once you receive this command prompt, replace the reported bad block by specifying the replace command. The rabads program provides on-line help information. You may use the rabads replace command to replace one bad block. You also may use the rabads init command to scan either the entire disk or a specified area and replace any bad blocks found. Once the bad blocks have been replaced, exit the rabads program by specifying the exit command. For further information, read Appendix G, Rabads Program Example.

3. Reboot your ULTRIX-11 system by executing your system's hardware boot ROM. For further information, read Section 3.3, Autobooting from the system disk.

NOTE

If a rabads replacement operation is interrupted (system crash or power failure), the disk may be left in an unusable state. When this occurs, you must reboot the rabads program and reissue the replacement command. The rabads program automatically provides instructions for restarting the interrupted replacement operation.

If the "Force Error Modifier" flag is set during a replacement operation, the bad block was replaced, but the rabads program could not verify that the data written was uncorrupted. Consequently, you must verify the data and clear the "Force Error Modifier" flag by rewriting the replacement block. To do so, use this procedure:

1. Shut down multiuser mode by using the opser program and specifying the s command. Once the system is in single-user mode, check all file systems on the system and user disks by specifying the f command. Escape the opser program by specifying the !sh command. For further information, read Chapter 5, ULTRIX-11 Operator Services.

2. Once you receive a shell prompt, determine the bad block's inode number by using the icheck command with the -b option specified. Then, determine the corresponding file name by using the ncheck command with the -i option specified. For further information, read icheck(1M) and ncheck(1M) in the ULTRIX-11 Programmer's Manual, Volume 1.

3. Mount the file system that contains the file by using the mount command. Once the file system is mounted, determine the amount of corruption by attempting to read through the file. If possible, correct corrupted data by restoring the file from backup media. For further information, read mount(1M) and restor(1M) in the ULTRIX-11 Programmer's Manual, Volume 1.

4. Once the data has been sufficiently restored, quit the shell and return to the opser program by pressing <CTRL/D>. Once receiving an opr> prompt, halt the processor by specifying the halt command.

5. Execute the system's hardware boot ROM and wait for the message:

   (<CTRL/C> aborts the auto-boot)

   Quickly, abort the autoboot procedure by pressing <CTRL/C>. Once you receive a boot prompt, type one of these commands:

   Boot: rc(1,0)/sas/rabads (RC25)

   Boot: ra(0,0)/sas/rabads (RA60/RA80/RA81)

   This command boots the stand-alone rabads program from the system disk. Once running, the rabads program prints this program banner and command prompt:

   ULTRIX-11 MSCP Disk Initialization Program

   rabads <help exit drives status table init replace>:

   Once you receive this command prompt, force the replacement block to be rewritten by specifying the replace command. This clears the "Force Error Modifier" in the replacement block. The rabads program provides on-line help information. Once the bad blocks have been replaced, exit the rabads program by specifying the exit command. For further information, read Appendix G, Rabads Program Example.

6. Reboot your ULTRIX-11 system by executing your system's hardware boot ROM. For further information, read Section 3.3, Autobooting from the System Disk.

4.4 Monitoring General System Activity

As system manager, you should monitor general system performance on a regular basis. You can use these ULTRIX-11 commands to monitor system performance:

ac	Reports login accounting information.
badstat	Reports disk bad block information.
bufstat	Reports I/O buffer cache usage.
iostat	Reports I/O transfer information.
memstat	Reports system memory usage.
ps	Reports active process information.
pstat	Reports system table information.
sa	Reports system accounting information.
who	Reports either the names of those users that currently are logged in or a login history.

In addition, the ULTRIX-11 error logging system captures information about device errors and saves it in the error log file. When a fatal error occurs, a message is printed on the console terminal, and the user process is notified with a fatal error return. When a recoverable error occurs, a message is entered in the error log file, but the user process is not notified. Because an increased soft error rate often precedes a hardware failure, you should examine the error log file on a regular basis.

For further information, read Chapter 8, ULTRIX-11 Error Logger as well as ac(1M), badstat(1M), bufstat(1M), iostat(1M), memstat(1M), ps(1), pstat(1M), sa(1M), and who(1) in the ULTRIX-11 Programmer's Manual, Volume 1.

4.5 Creating User Accounts

As system manager, you should use this procedure to create a new user account:

- Edit /etc/passwd file and create a login entry
- Create user's home directory
- Assign user's login password
- Create required shell start-up files

4.5.1 Editing the /etc/passwd File

For each new user, you must create a new entry in the /etc/passwd file. Each entry in /etc/passwd contains information that the system uses in verifying login permission and in establishing both the user's environment and initial process. Each entry has this format:

name:password:userID:groupID:finger:directory:shell

name	Contains the user's login name (maximum eight lowercase characters). The system uses this name when verifying login permission.
password	Contains the user's encrypted password, if used. The system uses this password in verifying login permission. When creating a new entry, leave this field blank. Later, you can assign the user's login password with the passwd command.
userID	Contains the user's unique user identification number. Although the user specifies a name when logging in, the system translates this name to this ID number and uses it both in identifying the user's processes and in determining owner access permission to files.
groupID	Contains the user's group identification number. The system uses this ID number in determining group access permission to files. To form user groups, use the newgrp command and /etc/group file. For further information, read newgrp(1) and group(5) in the ULTRIX-11 Programmer's Manual, Volume 1.
finger	Contains information that is used by the finger command. When creating a new entry, enter the user's full name. By using the chfn command, the user later may supply additional information.
directory	Contains the pathname to the user's home or login directory. The system uses this pathname

in placing the user in that directory during the login process. If you do not specify a directory pathname, the system automatically places that user at the root directory of the file system, /.

shell — Contains the pathname of the user's initial process. The system uses this pathname in invoking the designated program at the end of the login process. Although the pathname of any program may be specified, the initial process normally is a login shell (/bin/sh or /bin/csh). If a pathname is not specified, the system automatically invokes the Bourne shell (/bin/sh).

The following sample entries from /etc/passwd are for a demonstration account and two user accounts:

```
demo::10:20:Demo Account:/usr/demo:/usr/local/demo.program
smith::20:30:Jim Smith:/user/smith:
jones::21:30:Ed Jones:/user/jones:/bin/csh
```

The login name for the first entry is demo. A login password is not assigned. The assigned user and group IDs are 10 and 20, respectively. The assigned finger information is Demo Account. The assigned home directory is /usr/demo (/usr file system), and the assigned initial process is a demo program in /usr/local.

The login name for the second entry is smith. Again, a login password is not assigned. The assigned user and group IDs are 20 and 30, respectively. The assigned finger information is Jim Smith. The assigned home directory is /user/smith (/user file system), and the assigned initial process is the Bourne shell (default).

The login name for the third entry is jones. Again, a login password is not assigned. The assigned user and group IDs are 21 and 30, respectively. (Both Smith and Jones belong to the same group and, therefore, have group access permission to the same files.) The assigned finger information is Ed Jones. The assigned home directory is /user/jones (/user file system), and the assigned initial process is the C shell.

4.5.2 Creating a User Home Directory

Having created a new entry in /etc/passwd, create a directory in the specified file system and then change its owner to the appropriate user.

First, ensure that the file system where the user's home directory is to be located is mounted. If the system is in single-user mode, type this command sequence:

```
# /etc/mount -a
```

This command mounts all the file systems listed in the /etc/fstab file. If the system is in multiuser mode already, these file systems should have been mounted during the transition from single-user to multiuser mode.

Then, create the user's home directory and change its assigned userID and groupID to that user. The following example lists the sequence of commands that would be entered for the three users cited above in the sample /etc/passwd file.

```
# cd /usr
# mkdir demo
# chog demo demo
# cd /user
# mkdir smith jones
# chog smith smith
# chog jones jones
# cd /
```

If you created user home directories while the system was in single-user mode, you should unmount all file systems before invoking multiuser mode. To do so, type this command sequence before pressing <CTRL/D>:

```
# /etc/umount -a
```

For further information, read chog(1) and mkdir(1) in the ULTRIX-11 Programmer's Manual, Volume 1.

4.5.3 Assigning a User Password

Having created the user's home directory, next assign a login password to each new account. For each new account, log in as that user (type the appropriate login name).

Once logged in as that user, use the passwd command to assign a login passwd (6 character minimum). The passwd command prompts for the password twice for verification. If your responses match, the passwd command assigns the new encrypted password to the /etc/passwd entry. If your responses do not match, the passwd command does not assign the password.

Before new users attempt to log in, you should tell them their assigned login password. Once logged in, they can change their login password by using the passwd command. For further information, read passwd(1) in the ULTRIX-11 Programmer's Manual, Volume 1.

4.5.4 Creating Shell Startup Files

Having assigned a password for each new account, next create the required shell start-up files. Upon executing, the login shell looks in the user's home directory for its appropriate start-up files and uses this information to set up the user's shell environment.

If the user is using the Bourne shell (/bin/sh), you should create a .profile in the user's home directory. If the user is using the C shell (/bin/csh), you should create a .cshrc and .login in the user's home directory. For further information, read csh(1) and sh(1) in the ULTRIX-11 Programmer's Manual, Volume 1. In addition, read An Introduction to the C Shell and An Introduction to the UNIX Shell in the ULTRIX-11 Programmer's Manual, Volume 2A.

4.6 Setting Up User File Systems

As system manager, you determine the quantity, size, and location of your user file systems. The user file system layout is based primarily on the number and type of disks available as well as the needs of your user community.

During a system generation, you configure user file systems on logical disk units. For nonpartitioned disks, a logical unit is a physical disk unit. For partitioned disks, a logical unit is one of the subunits (pseudodisks) on the physical disk unit. For further information, read Section 1.3, Logical Partitioning of Disks.

The following sections discuss the issues that you should consider when setting up user file systems.

4.6.1 Sysgen of User Disks

During a system generation, you should ensure that all disk units that are to contain file systems are configured into the ULTRIX-11 kernel. In addition, you have to create the special files for each disk unit.

Normally, these tasks are done during an ULTRIX-11 installation. If the disk units have not been configured into the kernel, however, read Chapter 2, ULTRIX-11 System Generation, and follow the procedure discussed in Section 2.6.1, Support Files for Disks.

4.6.2 Qualifying Disk Media

Before locating a user file system on any disk, you should ensure that the disk media is acceptable for ULTRIX-11 system use. Specifically, you have to determine the bad blocks on each disk. For a description of the disk media qualification procedure, read Appendix B, Disk Media Qualification Procedures, in the ULTRIX-11 Installation Guide.

4.6.3 Determining the Number of User File Systems

You should create a file system for each group of users that either works on a common project or must share files. If you use nonpartitioned disks, each physical unit can hold exactly one file system. Therefore, the number of file systems that you set up is limited to the number of disk units that you have available. If you use partitioned disks, however, each physical unit can hold more than one file system.

NOTE

For partitioned disks, DIGITAL recommends that you set up several smaller user file systems instead of one large, comprehensive user file system.

4.6.4 Determining the Size of User File Systems

For nonpartitioned disks, you should set the size of each user file system to that of the physical unit (disk size). If you set the size of a user file system smaller than that of its physical unit, you waste the remaining space on that physical disk unit.

For partitioned disks, you should set the size of each user file system to the size of the logical unit (partition size). If you set the size of a user file system smaller than that of its logical unit, you waste the remaining space.

For further information, read Appendix D, Disk Logical Partition Sizes.

NOTE

When setting the size of file systems on RC25, RD51/RD52, and RA60/RA80/RA81 disks, pay special attention to the number of blocks that are reserved for a maintenance area. To avoid overstepping this area when setting the sizes of your user file system on these disks, use the rasize command to determine the location and sizes available. For further information, read rasize(1M) in the ULTRIX-11 Programmer's Manual, Volume 1.

4.6.5 Determining the Location of User File Systems

For nonpartitioned disks, you can place the user file systems on any disk but the system disk. For systems on RL01s, units 0 and 1 are reserved for the system disk. For systems on RC25s, unit 1 normally is the system disk. For all other systems, unit 0 normally is the system disk.

For partitioned disks, the system normally resides on the first three logical partitions:

- Partition 0 for the root file system
- Partition 1 for the swap area
- Partition 2 for the /usr file system

Therefore, for partitioned disks, you should locate the user file systems either on a remaining system disk partition or on any desired partition of a nonsystem disk unit. For further information on locating your user file systems, read Appendix D, Disk Logical Partition Sizes.

NOTE

Partitioned disks have several different configurations, some of which overlap. Do create separate file systems on overlapping partitions.

4.6.6 Making File Systems

As system manager, you should create the ULTRIX-11 file system structure on each logical disk unit that holds a file system. To make a file system on an unused disk partition, use the mkfs command. In response to the superuser prompt, type this command sequence:

```
# /etc/mkfs /dev/rxxnp size disk cpu fsname volname
```

xx — Specifies the ULTRIX-11 logical disk name. For a list of logical names, read Appendix C, ULTRIX-11 Device Names and Major Device Numbers.

n — Specifies the disk unit number.

p — Specifies the disk partition number.

size — Specifies the size of the partition in blocks. For further information, read Appendix D, Disk Logical Partition Sizes.

disk — Specifies the generic disk name.

cpu — Specifies the last two digits of the processor type.

fsname — Specifies the name (6-character maximum) that is to be recorded in the file system's superblock.

volname — Specifies the volume label (6-character maximum) that is to be recorded in the file system's superblock.

The following command sequence creates a file system on an RL02 disk, unit 1, and a PDP-11/24:

```
/etc/mkfs /dev/rrl17 20480 rl02 24 user usrdsk
```

The mkfs command uses the generic disk name and processor type in determining the proper file system interleave factors to optimize I/O throughput. For further information, read mkfs(1M) in the ULTRIX-11 Programmer's Manual, Volume 1.

CAUTION

When you use the mkfs command to make a new file system on a disk partition that already contains a file system, you destroy all the existing data on that file system. Therefore, when making a new file system, either verify that the disk partition is unused or dump the existing data to tape before running the mkfs command. To verify that a disk partition does not already contain a file system, use the fsck command.

4.6.7 Mounting and Unmounting File Systems

To mount a file system, use the mount command. In response to the superuser prompt, type this command sequence:

```
# /etc/mount special directory
```

special — Specifies the block-mode special file name for the logical unit that contains the file system.

directory — Specifies the directory name on which the file system is to be mounted.

When you run the mount command without any arguments, the system lists those file systems that currently are mounted. The following example mounts a file system contained on an RL02, unit one, on the /user directory.

```
# /etc/mount /dev/rl17 /user
```

To unmount a file system, use the umount command. In response to the superuser prompt, type this command sequence:

```
# /etc/umount special
```

special — Specifies the block-mode special file name for the logical unit that contains the file system.

The following example unmount a file system contained on an RL02, unit one:

```
# /etc/umount /dev/rl17
```

For further information, read mount(1M) and mtab(5) in the ULTRIX-11 Programmer's Manual, Volume 1.

4.6.8 Editing the /etc/fstab File

The /etc/fstab file contains information that is required by all system programs and commands that access file systems by name. Each entry in /etc/fstab contains information in the following format that describes a file system:

special:directory:mode

special — Specifies the block-mode special file name for the logical unit that contains the file system.

directory — Specifies the directory name on which the file system is to be mounted.

mode — Specifies how to mount the file system. For example, rw indicates read/write access; ro indicates read only status; and xx indicates ignore this entry.

The following are sample /etc/fstab entries for four file systems:

```
/dev/hp00:/:rw
/dev/ml0:/tmp:rw
/dev/hp03:/usr:rw
/dev/hp22:/archive:ro
```

This file system table includes the root file system (hp00) mounted read/write, the /tmp file system (ml0) mounted read/write, the /usr file system (hp03) mounted read/write, and the /archive file system (hp22) mounted read only.

When you use the mount and umount commands with the -a option, they read /etc/fstab and either mount or unmount all listed file systems. When the -a option is specified, the mount command reads and processes the entries in /etc/fstab in the order that they appear (first to last). Conversely, when the -a option is specified, the umount command reads and processes the entries in the reverse order (last to first). Therefore, the order in which you specify the /etc/fstab entries is important. For the mount and umount

programs to nest properly, the entries must appear in a logically correct sequence.

The following sample /etc/fstab table indicates that the /usr file system is to be mounted before the /usr/staff file system is mounted on it and that the /usr/staff file system is to be unmounted before the /usr file system is unmounted:

```
/dev/hp00:/:rw
     .
     .
     .
/dev/hp13:/usr:rw
/dev/hp12:/usr/staff:rw
```

This sample is in the logically correct sequence. If the order of the last two entries were reversed, however, both the mount and umount commands would fail.

NOTE

Normally, the multiuser start-up file, /etc/rc, automatically mounts all file systems listed in the /etc/fstab file during the transition from single-user mode to multiuser mode. For further information, read Section 1.2.4, Multiuser Mode.

4.7 Enabling User Terminals

To make a terminal line available for interactive use, you should:

- Configure the device driver into the kernel
- Create the required special file
- Verify the TTY structure assignment
- Enter terminal type in the /etc/ttytype file
- Enable terminal line in the /etc/ttys file

4.7.1 Configuring Communications Device Drivers

The device driver for the terminal line's communications interface (for example, KL11, DL11, DH11, DHU11, DZ11, DHV11, DZQ11, DZV11) must be configured into the ULTRIX-11 kernel. If the driver is not already configured into the kernel, you should:

- Run the sysgen program
- Create a new configuration file
- Make a new kernel
- Install and boot the new kernel

For more specific information, read Chapter 2, ULTRIX-11 System Generation.

4.7.2 Creating Communications Interface Special Files

Using both a port on the communications interface and the device driver, the communications device special files let the operating system access terminal lines. In turn, each terminal line uses both a port on the communications interface and the device driver. If the required special files do not already exist, you must create them.

For detailed information about communications special files, read Section 2.6.3, Support Files for TTY Interfaces, and Section 1.4 in the ULTRIX-11 Installation Guide.

4.7.3 Verifying TTY Structures

The ULTRIX-11 kernel contains a pool of TTY data structures. Each TTY structure provides information that the operating system requires to support any communications interface port that uses a general terminal interface.

Since the boot program automatically assigns a TTY structure to each port during system initialization, you do not need to assign TTY structures. To list the assigned TTY structures, use the tss command. In response to the shell prompt, type:

```
$ /etc/tss
```

The output from this command can be helpful when debugging a user-written device driver that uses the general terminal interface.

4.7.4 Editing the /etc/ttytype File

The /etc/ttytype file identifies the terminal type that is connected to a communications interface port. Specifically, this file associates the generic name of the terminal with its logical, special file name for the communications interface port.

The /etc/ttytype file has this format:

```
type  tty##
```

type Specifies the terminal type.

tty## Specifies the special file name for the communications interface port.

For example, the following are sample entries from the /etc/ttytype file:

```
la120   console
vt100   tty00
vt52    tty01
dialup  ttyd0
```

When a user logs in, the login program reads the /etc/ttytype file and sets the user's TERM environmental variable to the designated terminal type. For example, using the entry for tty01 above, the login program sets:

```
TERM = vt100
```

If the terminal type is not defined in /etc/ttytype, the login program sets the TERM variable to type "unknown."

To operate properly, some programs (for example, the vi editor) require that the TERM variable be set. Therefore, you should edit /etc/ttytype and add an entry that identifies the type of each newly configured terminal. For further information about environmental variables, read environ(5) in the ULTRIX-11 Programmer's Manual, Volume 1.

4.7.5 Editing the /etc/ttys File

The operating system uses the /etc/ttys file both to enable and disable terminal lines as well as to set terminal characteristics. Each entry in /etc/ttys lists information for one terminal line and has this format:

nctty##

n Specifies whether the terminal is to be enabled (remote, local, or no login) or disabled.

c Specifies the terminal's characteristics.

tty## Specifies the special file name for the communications interface port.

The possible values for n are:

```
0 = Disabled  (ignored during initialization)
1 = Remote    (enabled for dialup access)
2 = Local     (enabled for local access)
3 = No Login  (enabled for tip/uucp access)
```

Some of the possible values for c are:

```
0 = cycles thru 300-1200-150-110 bits/sec
2 = 9600 bits/sec
3 = 1200 cycles back to 300 bits/sec
6 = 2400 bits/sec
7 = 4800 bits/sec
f = 1200 bits/sec
```

For further information, read gettytab(5) and getty(8) in the ULTRIX-11 Programmer's Manual, Volume 1.

The following are five sample entries from the /etc/ttys file:

```
24console
00tty00    (disabled)
22tty01    (9600 bits/sec local)
30tty02    (tip/uucp access only)
13ttyd0    (1200/300 bits/sec dialup)
```

NOTE

The first line of the /etc/ttys file always enables the console terminal. You should never change this line.

To enable or disable terminals or to change terminal characteristics, use one of the ULTRIX-11 editors or the ted program. To modify the /etc/ttys file with the ted program, type:

```
# ted
```

The ted program has on-line help messages that are intended to assist you. Once the ted program is running, type this command sequence to obtain help:

```
help ted
```

For further information, read ted(1) in the ULTRIX-11 Programmer's Manual, Volume 1.

You can modify entries in /etc/ttys while the system is in single-user mode or multiuser mode. However, the procedures that you use differ.

When the system is in single-user mode, use either one of the ULTRIX-11 editors or the ted program to modify /etc/ttys. Then, during the transition from single-user mode to multiuser mode, the system automatically implements the changes.

When the system is in multiuser mode, DIGITAL recommends that you use the ted program to modify /etc/ttys. The ted program does not allow you to change the entry for either the system console or your own terminal. Then, when you write back the contents of /etc/ttys, ted automatically implements the changes.

If you do use one of the ULTRIX-11 editors while in multiuser mode, you first should verify the change and then write back the contents of /etc/ttys. Then, send a HANGUP signal to the system's initialization process, /etc/init. To send a HANGUP signal to the init process, type this command sequence:

```
# kill -1 1
```

This command causes the init process to reread /etc/ttys and implement the changes.

For further information about the general terminal interface and the /etc/ttys file, read tty(4), gettytab(5), tty(5), ttytype(5), getty(8), and init(8) in the ULTRIX-11 Programmer's Manual, Volume 1.

4.8 Setting Up the cu Facility

Although the cu facility is obsolete and DIGITAL recommends that you use the tip facility in its place, the following section discusses the procedure for setting up the cu facility.

To set up the cu facility, you should first read cu_v7m(1C) in the ULTRIX-11 Programmer's Manual, Volume 1 and then use this procedure:

1. Set up the cu software
2. Select the cu hardware
3. Connect the cu hardware
4. Verify cu operations

4.8.1 Setting Up the cu Software

For the Micro/PDP-11, the cu programs are optional software. To load the cu software, use the distributed osload program. In response to the superuser prompt, type this command sequence:

```
# osload load orphan
```

This command loads not only the cu software but also some extra, unrelated files. To remove these extra files, type this command sequence:

```
# rm -f /bin/bas /bin/*.v7_NS
```

Next, with the system in single-user mode, install the cu programs. In response to the superuser prompts, type this sequence of commands:

```
# cd /
# /etc/mount -a
# rm -f /usr/bin/cu
# cd /bin
# mv cu_v7m cu
# mv custat_v7m custat
# chmod 755 cu custat
# chog bin cu custat
# cd /
# /etc/umount -a
# sync
```

You also should ensure that the device driver for the cu communications interface ports is configured into the ULTRIX-11 kernel. For further information, read Chapter 2, ULTRIX-11 System Generation.

With the system still in single-user mode, create the special files for the cu ports (cua# for ACUs or cul# for

direct lines). In response to the superuser prompt, type this sequence of commands:

```
# cd /dev
# ln tty## cua#
# ln tty## cul#
# chmod 666 cu??
# cd /
# sync
```

tty## Specifies the name of the special file for the communications line assigned to the cu port.

Specifies the number of the cu port (for example, 0, 1, 2). The cu special files should have mode 0666 (rw-rw-rw-).

Finally, edit the /etc/ttys file and enable each of the cu ports for NO LOGIN (for example, 30tty##). To edit the /etc/ttys file, DIGITAL recommends that you use the ted command. For further information, read Section 4.7, Enabling User Terminals.

4.8.2 Selecting the cu Hardware

The cu facility supports three types of connections:

- Hardwired direct connection
- Direct connection by modem
- Phone access by autocall modem

The hardwired direct connection requires a null modem cable (for example, BC03-M). As system manager, you should choose the hardware type for modem direct connection. For direct connection by modem, the modems are connected to the cu ports with a straight-through cable (for example, BC05-D). For phone access by autocall modem, the cu facility supports two autocall units:

- DF02-AC (300 bits/sec)
- DF03-AC (300/1200 bit/sec)

Although the DF03-AC operates at 300 or 1200 bits/sec, it cannot switch speeds automatically. Therefore, you should select the speed with the HS switch on the DF03-AC front panel. Then, the communications device that you use to drive the cu ports must have modem control. DIGITAL recommends that you use one of the following:

- DL11-E or DLV11-E
- DZ11 or DZV11 or DZQ11
- DHU11 or DHV11
- DH11 (with DM11-BB modem control option)

4.8.3 Connecting the cu Hardware

To install a hardwired direct connection, connect (with a null modem cable) the cu port on the local system to an available port on the remote system.

To install a direct connection with a modem, connect (with a straight-through cable) the cu port on the local system to an available port on the remote system. Follow the modem installation instructions to set up a direct modem link.

To install a phone connection with an autocall modem, connect (with a straight-through cable) the DF02-AC or DF03-AC unit to the cu port on the local system. Then, connect the ACU to the phone line by following the instructions in the DF02-AC or DF03-AC User's Guide. Finally, set the ACU communications bit rate. For more specific information, refer to the "switch options on the DF02-AC or DF03-AC automatic call unit" in the appropriate user's guide. If the ACU is connected to a DL11/DLV11 interface, the ACU communications bit rate must match the DL11 speed. For interfaces with programmable speeds (DH11/DHU11/DHV11 or DZ11/DZV11/DZQ11), set the ACU communications bit rate to 300 bits/sec.

4.8.4 Verifying cu Operations

To test a hardwired direct connection, type this command sequence:

```
# cu -t -s #
```

Specifies the speed (for example, 9600 bits/sec).

When the connection is established, log in to the remote system.

To test a direct connection by modem, type this command sequence:

```
# cu -t -m -s #
```

Specifies the speed (for example, 1200 bits/sec).

When the connection is established, log in to the remote system.

To test phone access by autocall modem, type this command sequence:

```
# cu telno -s #
```

telno Specifies the remote system's telephone number.

Specifies the speed (for example, 1200 bits/sec).

If the speed is incorrect, type ~# and press the <RETURN> key to step to the next speed. Once the connection is established, log in to the remote system. To break the connection after logging out, type ~. and press the <RETURN> key.

NOTE

To select an available ACU or line, use the custat command. For further information, read cu_v7m(1C) in the ULTRIX-11 Programmer's Manual, Volume 1.

4.9 Setting Up tip Connections

To set up the tip facility, you should first read tip(1C), remote(5), and phones(5) in the ULTRIX-11 Programmer's Manual, Volume 1 and then use this procedure:

1. Set up the tip software
2. Select the tip hardware
3. Connect the tip hardware
4. Verify tip operations

4.9.1 Setting Up the tip Software

For the Micro/PDP-11, the tip programs are optional software. To load the tip software, use the distributed osload program. In response to the superuser prompt, type this command sequence:

```
# osload load tip
```

You also should ensure that the device driver for the tip communications interface ports is configured into the ULTRIX-11 kernel. For further information, read Chapter 2, ULTRIX-11 System Generation.

Next, to ensure that the special files for the tip ports exist and that their mode is 0666, type this sequence of commands:

```
# cd /dev
# ls
# chmod 666 tty## tty## tty##
# cd /
# sync
```

tty## Specifies the name of the special file for the communications line assigned to the tip port.

Next, edit the /etc/ttys file and enable each tip port for NO LOGIN (for example, 30tty##). To edit the /etc/ttys file, DIGITAL recommends that you use the ted command.

Finally, edit the /etc/remote file and type the required information for each remote system connection. The prototype remote file, /etc/remote, contains examples of how to format this information. For further information, read remote(5) in the ULTRIX-11 Programmer's Manual, Volume 1.

4.9.2 Selecting the tip Hardware

The tip facility supports three type of connections:

- Hardwired direct connection
- Direct connection by modem
- Phone access by autocall modem

The hardwired direct connection requires a null modem cable (for example, BC03-M). As system manager, you should choose the hardware type for modem direct connection. For direct connection by modem, the modems are connected to the tip ports with a straight-through cable (for example, BC05-D). For phone access by autocall modem, the tip facility supports two autocall units:

- DF02-AC (300 bits/sec)
- DF03-AC (300/1200 bits/sec)

For a list of known but unsupported devices, read through the /etc/remote file.

Although the DF03-AC operates at 300 or 1200 bits/sec, it cannot switch speeds automatically. Therefore, you should select the speed with the HS switch on the DF03-AC front panel. Then, the communications device that you use to drive the tip ports must have modem control. DIGITAL recommends that you use one of the following:

- DL11-E or DLV11-E
- DZ11 or DZV11 or DZQ11
- DHU11 or DHV11
- DH11 (with DM11-BB modem control option)

4.9.3 Connecting the tip Hardware

To install a hardwired direct connection, use a null modem cable and connect the tip port on the local system to an available port on the remote system.

To install a direct connection with a modem, use a straight-through cable and connect the tip port on the local system to an available port on the remote system. Follow the modem installation instructions to set up a direct modem link.

To install a phone connection with an autocall modem, use a straight-through cable and connect the DF02-AC or DF03-AC unit to the tip port on the local system. Then, connect the ACU to the phone line by following the instructions in the DF02-AC or DF03-AC User's Guide. Finally, set the ACU communications bit rate. For more specific information, refer to the "switch options on the DF02-AC or DF03-AC automatic call unit" in the appropriate User's Guide. If the ACU is connected to a DL11/DLV11 interface, the ACU communications bit rate must match the DL11 speed. For interfaces with programmable speeds (DH11/DHU11/DHV11 or DZ11/DZV11/DZQ11),

set the ACU communications bit rate to 300 bits/sec.

4.9.4 Verifying tip Operations

To test either a hardwired or a modem direct connection, type this command sequence:

```
# tip -# system
```

\# Specifies the speed (for example, 9600 bits/sec).

system Specifies the name of the remote system (from /etc/remote file).

Once the connection is established, log in to the remote system. To break the connection after logging out, type ~. and press the <RETURN> key.

To test phone access by an autocall modem, type this command sequence:

```
# tip -# phone
```

\# Specifies the speed (for example, 1200 bits/sec).

phone Specifies the remote system's telephone number.

If the speed is incorrect, type ~# and press the <RETURN> key to step to the next speed. Once the connection is established, log in to the remote system. To break the connection after logging out, type ~. and press the <RETURN> key.

NOTE

The tip facility supports the cu user interface and, therefore, accepts cu commands. You also may test these cu commands under the tip interface.

4.10 Installing the uucp Facility

For a detailed description of the uucp installation procedure, read UUCP Installation and Administration in the ULTRIX-11 Programmer's Manual, Volume 2B.

Chapter 5

ULTRIX-11 Operator Services

The opser program provides a simple and concise interface to ULTRIX-11 system maintenance. Specifically, the opser program enables a system operator who has only a basic understanding of the ULTRIX-11 system to:

- Obtain on-line help
- Determine who is currently logged in
- Shut down multiuser mode
- Check file system consistency
- Back up file systems
- Escape to shell and execute an ULTRIX-11 command
- Restart multiuser mode
- Halt the processor

The remaining sections of this chapter provide discussions of these tasks.

5.1 Running the opser Program

To run the opser program, the system operator should log in to the operator account (operator login name). For the operator account, the system automatically invokes the opser program (/opr/opser) in place of a shell.

Once running, the opser program prints a program header, two informational messages, and a command prompt:

```
ULTRIX-11 Operator Services

To correct typing mistakes:

        <DELETE> erases the last character,
        <CTRL/U> erases the entire line.

For help, type h then press <RETURN>

opr>
```

The opr> prompt is to remind the operator both that the opser program is running in place of the shell and that it is ready to accept opser commands.

To use an opser command, type the appropriate command letter or name and press the <RETURN> key. For example, to receive on-line help about all opser commands, type:

```
opr> h
```

The opser program then displays the following information:

```
() - may use first letter in place of full name
Valid commands are:

!sh             - shell escape (execute ULTRIX-11 commands)
                  (Type <CTRL/D> to return from shell)
(u)sers         - show logged in users
(s)hutdown      - stop time-sharing
(f)sck          - file system checks
(r)estart       - restart time-sharing
(h)elp          - print this help message
backup cfn      - file system backup
                  (cfn = command file name)
halt            - halt processor
^D (<CTRL/D>)   - exit from opser
```

To end an opser session and return to a login prompt, the operator should press <CTRL/D>.

NOTE

The operator can run the opser program from either the system console or a terminal. If the opser program is run from the console, the operator can use the full set of opser commands. If it is run from a terminal, the operator can use the h (help) and u (users) commands only.

The next seven sections provide more detailed discussions of the opser commands. For further information, read Appendix E, Opser Program Example.

5.1.1 Determining Who Is Currently Logged In

To determine how many users currently are logged in, use the opser program and specify the u command. In response to the opr> prompt, type:

```
opr> u
```

When the users command is specified, the opser program first displays a list of users who currently are logged in. On receiving the opr> prompt, the operator can use other opser commands.

5.1.2 Shutting Down Multiuser Mode

To shut down multiuser mode and leave the system in single-user mode, use the opser program and specify the s (shutdown) command. In response to the opr> prompt, type:

```
opr> s
```

When the shutdown command starts executing, the opser program first displays the names of those users that currently are logged on the system. Then, it prompts for the number of minutes to delay before shutting down multiuser mode. This delay is to give users enough time to finish their work and log out. During this delay period, the opser program broadcasts warnings of the impending shutdown at 1-minute intervals and disables further logins. Finally, at the designated time, it broadcasts a final message and then shuts down multiuser mode.

During the transition back to single-user mode from multiuser mode, the opser program kills all running processes and unmounts all mounted file systems. On receiving the opr> prompt, the operator is automatically in a single-user

environment. Then, the operator can use other opser commands and continue with system maintenance.

The operator can cancel the shutdown command at any time before the final shutdown message. To cancel a scheduled shutdown, press <CTRL/C>.

5.1.3 Checking File System Consistency

To check the consistency of your file systems when running the opser program in single-user mode, specify the f (file system check) command. In response to the opr> prompt, type:

```
opr> f
```

When the f command is specified, the opser program invokes the fsck program to check the file systems that are named in /etc/fstab. For further information, read Section 4.1.2, Check File System Consistency, and Section 4.1.3, Correct File System Inconsistencies.

NOTE

DIGITAL recommends that you check all file systems before backing up file systems. To shut down multiuser mode prior to checking file systems, specify the s (shutdown) command. For further information, read Section 5.1.2, Shutting Down Multiuser Mode.

5.1.4 Backing Up File Systems

To back up file systems when running the opser program in single-user mode, specify the backup command. In response to the opr> prompt, type this command sequence:

```
opr> backup cfn
```

The cfn argument specifies the command file name (daily, weekly, or monthly) that contains the ULTRIX-11 commands that are required for that backup. The system manager is responsible for setting up the backup command files and for notifying the operator of their names and schedule. For further information, read Section 4.2, Backing Up and Restoring File Systems.

NOTE

DIGITAL recommends that you check all file systems before backing up file systems. To shut down multiuser mode prior to checking file systems, specify the s (shutdown) command. For further information, read Section 5.1.2, Shutting Down Multiuser Mode.

5.1.5 Escaping to the Shell

To escape to the shell at any time when running the opser program, specify the !sh (escape shell) command. In response to the opr> prompt, type:

```
opr> !sh
```

After receiving a shell prompt, the operator can execute any ULTRIX-11 command. This command is useful for using the wall or write commands to notify users of an impending shutdown. Having finished with the desired ULTRIX-11 commands, press <CTRL/D> to return to the opser program. On receiving an opr> prompt, the operator can use other opser commands.

NOTE

When the operator escapes from the opser program, the shell comes up with superuser privilege. As system manager, therefore, you should permit the operator to use only a limited set of ULTRIX-11 commands.

5.1.6 Restarting Multiuser Mode

To restart multiuser mode when running the opser program in single-user mode, specify the r (restart) command. In response to the opr> prompt, type:

```
opr> r
```

Essentially, this command causes the system to read the /opr/restart file (roughly equivalent to /etc/rc) and restarts multiuser mode without having to reboot. For further information, read Section 1.2.4, Multiuser Mode.

5.1.7 Halting the Processor

To halt the processor when running the opser program in single-user mode, specify the halt command. In response to the opr> prompt, type this command:

```
opr> halt
```

This command prompts the operator for confirmation, delays for one second, and then halts the processor.

NOTE

To shut down multiuser mode prior to halting the processor, specify the s (shutdown) command. For further information, read Section 5.1.2, Shutting Down Multiuser Mode.

Chapter 6

ULTRIX-11 Text Overlay Scheme

To run large programs on your ULTRIX-11 system, you must use this text overlay scheme. Although the ld loader automatically does much of the work for creating program overlays, you must decide which module is to go into what overlay. This chapter provides information that is intended to help you in making this decision.

The remaining sections of this chapter discuss:

- Why program overlays are needed
- What should go into an overlay
- What is the proper command sequence
- How the ld loader creates overlaid programs
- How the C stack frame is used

6.1 Why Program Overlays?

On split I and D processors, programs must use overlays when their instructions cannot fit into 64K. Overlays on split I and D programs do not affect the size of a program's data segment. All split I and D programs have 56K of usable data space: 64K minus 8K required for the stack.

On nonsplit I and D processors, programs must use overlays when their instructions and data cannot fit into 56K. Overlays on nonsplit I and D programs, however, do affect the size of a program's data segment. All overlaid nonsplit I and D programs have 40K of usable data space: 64K minus 8K each for the stack, the base text segment, and the overlay segment. The text space saved by using overlays can then be used as additional data space.

For example, consider a program with 48K of text. If overlays were not used, this program would only have 8K for data: 64K minus 8K for the stack and another 48K for the text. If overlays were used, however, the saved text space then could be used to expand the data space. If an 8K base text segment and five 8K overlays were created, the remaining 32K of text space could then be used to increase the data space to 40K.

6.2 What Should Go into an Overlay?

When deciding what goes into an overlay, you should consider:

- Each module's size
- What other modules each calls

If possible, functions that repetitively call each other should be placed into the same overlay. Because no overlay switching is required to call functions in the base text segment, functions called frequently from other overlays should be put into the base text segment.

Figure 6-1 illustrates the logical layout of an overlaid nonsplit I and D program:

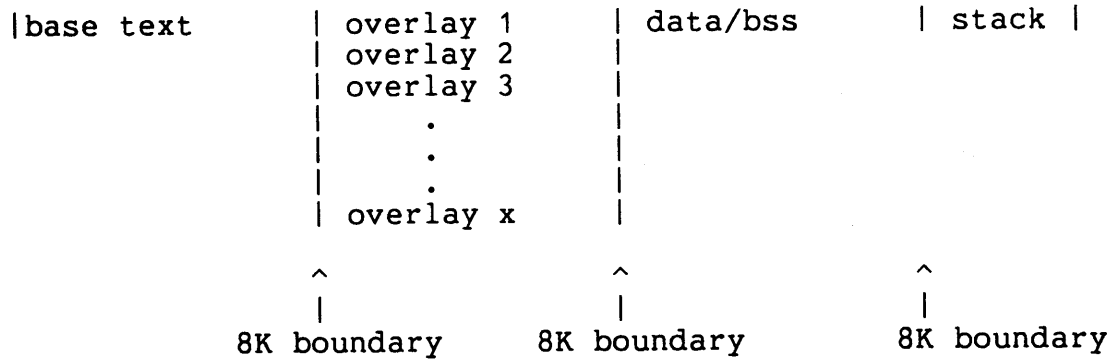


Figure 6-1 -- Overlaid Process

The starting address of the base text segment is 0. Because of hardware constraints, the starting address of the overlay segment is the size of the base text segment rounded up to the nearest 8K. The starting address of the data/bss segment is the starting address of the overlay segment plus the size of the largest overlay rounded up to the nearest 8K boundary. These boundaries are set automatically by the ld loader when either the -i or -n flag is specified.

For example, consider a program whose base text segment is 10K. Because it is rounded up to the nearest 8K, you can put more functions into it to fill it to 16K and not increase the base text segment. Similarly, consider a program that has three overlays (14K, 7K, and 8K) and whose first overlay must be at least 14K. You could combine the second and third overlays (7K + 8K). This would leave you with two overlays (14K and 15K) but would not increase your usable data space. If the first overlay could be split in two, however, you then could create four overlays (7K, 7K, 8K, and 7K) and increase your usable data space by 8K.

6.3 How Do You Create Overlays?

To specify overlays in the cc or f77 command line, precede the appropriate module with the -Z flag. Each successive -Z flag in the command tells the loader that, beginning with the next module, switch to the next overlay. To end overlays, specify the -L flag in the command line. The remaining modules specified on the command line then are to go into the base segment.

For example, consider a C program consisting of the following seven modules: a.c, b.c, c.c, d.c, e.c, f.c, and g.c. To compile them into object format, type:

```
cc -c a.c b.c c.c d.c e.c f.c g.c
```

Then, to determine the size of each object file, type:

```
size a.o b.o c.o d.o e.o f.o g.o
```

Based on this information, you decide to put a.c in the base, b.c and c.c in the first overlay, d.c in the second overlay, e.c and f.c in the third overlay, and g.c in the base segment. Then, to link the object files together, type:

```
cc -n a.o -Z b.o c.o -Z d.o -Z e.o f.o -L g.o
```

The -n flag indicates that the overlaid programs are to be shared text. If the -i flag is specified for a split I and D process, then the -n flag is redundant. You must, however, specify at least one of these flags. The -Z and -L flags are passed from cc to ld with their position relative to objects intact.

If a library should be referenced within an overlay on the cc command line, everything removed from that library goes into that overlay. For example, the following command sequence causes everything in the -ltermlib to be placed in an overlay:

```
cc a.o -Z b.o -Z -ltermlib -L
```

NOTE

Because they contain assembly routines that do not use csv/cret routines, you should not include the -lfpsim, -lc, or -ljobs libraries within overlays.

6.3.1 Creating an Overlaid Version of yacc

The source for the yacc program consists of: yla.c, ylb.c, y2.c, y3.c, and y4.c. To create an overlaid version of the yacc program, first compile the source into object format. To do so, type:

```
cc -c -O yla.c ylb.c y2.c y3.c y4.c
```

After creating the object files, use the size command to determine their sizes. To do so, type:

```
size yla.o ylb.o y2.o y3.o y4.o
```

The size command then prints the sizes of each object file:

```
yla.o: 3932+878+1000 = 5810b = 013262b
ylb.o: 834+134+0 = 968b = 01710b
y2.o: 6922+1974+2 = 8898b = 021302b
y3.o: 3012+536+0 = 3548b = 06734b
y4.o: 2142+636+0 = 2778b = 05332b
```

By reviewing the first number for each module (text size), you can determine that y2.o has a text size of 6922, which is close to the first 8K boundary. You also can determine that, when added together, ylb.o, y3.o, and y4.o have a text size of 5988, which is well within the first 8K boundary. In addition to these reported sizes, you also should consider that yla.c contains functions that are called repeatedly.

Based on this information, you determine that yla.o should go into the base text segment; that y2.o should go into the first overlay; and that ylb.o, y3.o, and y4.o should go into the second overlay. To link them together, therefore, type:

```
cc -o yacc40 -n yla.o -Z y2.o -Z ylb.o y3.o y4.o -L
```

If you use the size command to determine the sizes of this output, it prints:

```
7488+(6976,6016)+4566+36106 = 48160b = 0136040b (20480 total text)
```

Because library functions were brought in and thunks were created, the base text segment is much larger than the size originally reported for yla.o alone (3932). For information about thunks, read Section 6.5, Overlaid Programs and Thunks. All overlays are within the first 8K boundary, but their reported sizes show increases. Because of hardware constraints, the size of each was rounded up to the nearest 64 byte boundary: 6922 to 6976 and 5988 to 6016.

6.4 The C Stack Frame

Because the f77 and cc compliers use the same back end, the underlying function-calling procedure for code produced by both is the same.

The first assembly instruction upon entering a function is:

```
jsr r5,csv
```

Calling csv creates this stack frame:

```
          (high mem)
             ^
             |
          return pc
r5->      saved r5
          saved ovno
          saved r4
          saved r3
          saved r2
sp->
```

The next assembly instruction usually is:

```
sub     xx,sp
```

This instruction allocates space for local variables. These variables then are referenced as offsets from r5: -10.(r5) is the first, -12.(r5) the second, and so on.

Because they are later used to return values, the initial values of r0 and r1 are not saved. Short int values are stored in r0, while long int values are stored in r0-r1. The most significant is r0, while the least significant is r1.

This assembly instruction is used to return from a function:

```
jmp cret
```

The cret code compares the saved overlay number with the global variable _ovno (in assembly, __ovno). If they differ, it puts the saved overlay number into _ovno and r0. To switch the overlays, it then executes an EMT trap.

NOTE

If a program accidentally modifies either the _ovno variable or the saved overlay number, problems can result. On a nonoverlaid program, an EMT trap can occur. On an overlaid program, the wrong overlay may be switched.

Once the correct overlay is switched, the cret code restores the saved values of r2, r3, and r4. Then, it puts the current value of r5 into the stack pointer and pops a new r5 value off the stack.

The next assembly instruction, which finishes up the return, is:

```
ret pc
```

The functions csv and cret, which can be found in object format in the archive /lib/libc.a, determine exactly what the stack frame looks like.

This stack frame is not compatible with the standard forms of csv and cret found in other PDP-11 versions of UNIX (most notably V7). To allow for V7 compatibility, both the f77 and cc compilers provide a -V7 switch. This switch serves two purposes:

- When compiling object modules, local variables start at 8.(r5) instead of 10.(r5). All offsets in assembly code, therefore, must be adjusted accordingly.
- When calling the loader, /lib/v7csv.o is included before /lib/libc.a is searched. The earlier versions of csv/cret found in /lib/libc.a are not used.

These modified versions of csv/cret create this stack frame:

```
          (high mem)
             ^
             |
          save pc
r5 ->     saved r5
          saved r4
          saved r3
          saved r2
          blank word
sp ->
```

The blank word allows this version of csv/cret to work with both modules that have been compiled with and without the -V7 option. Even when the -V7 option is specified, functions, which start their local variables at 10.(r5), are also included from the standard libraries.

NOTE

Because the overlay number is not saved, the -V7 switch cannot be used on overlaid programs. The only time that the -V7 switch would be needed is when you need to link in modules that were compiled with compilers that start their local variables at 8.(r5).

6.5 Overlaid Programs and Thunks

The ld loader actually places a module into an overlay. When it does so, it renames the _foo global text labels ~foo. For each text label, it then creates these lines of assembly code, called a thunk, to become the interface to ~foo:

```
_foo:   mov $~foo+04,r1
        jmp ovhndlrX
```

The X indicates the overlay number into which ~foo has been placed. All thunks are eight bytes long and are placed into the base text segment. The following is sample ovhndlrX code:

```
ovhndlrX: mov $X,r0
          jmp ovhndlr
```

The ovhndlr is the general overlay handling routine. It first compares the current overlay number with that saved in r0. If they are different, it changes the overlay to point to the new one. A jsr r5,csv instruction is simulated. Then, it jumps to the location specified in r1. For this reason, all functions that go into overlays must have as their first instruction:

```
jsr r5,csv
```

In addition, they must return by:

```
jmp cret
```

This ensures that all overlay switching is done properly. Although this is done automatically by both the C compiler and the F77 compiler, you should follow these conventions in those assembly language routines that you place into overlays.

Chapter 7

ULTRIX-11 User-Mode System Exerciser Package

By using the system exerciser control program (sysx) and one or more exerciser modules, the ULTRIX-11 system exerciser package exercises your system hardware. Specifically, exerciser modules exist for:

- Processor
- Main memory
- Floating point
- Disk drives
- Tape drives
- Communications interfaces
- Line printers

Normally, exerciser modules run under the control of the sysx program. The system manager or DIGITAL field service representative uses the sysx program to:

- Create an exerciser script
- Start and stop the modules
- Monitor operations
- Print the error log files

The remaining sections of this chapter provide discussions of these sysx program functions, of each exerciser module, and of the exerciser options.

7.1 Running the System Exerciser Control Program

To start an exerciser session, you first must log in as the superuser (root account). The system then prints these messages and the superuser prompt:

```
Welcome to the ULTRIX-11 System

erase = delete, kill = ^U, intr = ^C
#
```

Once logged in, use the cd command to change directories to /sysxr. In response to the superuser prompt, type this command sequence:

```
# cd /sysxr
```

Once in the /sysxr directory, you are ready to use the sysx program.

You can run an exerciser module either individually or in conjunction with the sysx program. DIGITAL recommends that you should run all exerciser modules under the control of the sysx program. When run individually, exerciser modules ignore interrupts and cannot be stopped by pressing <CTRL/C>.

To use the sysx program, type:

```
# sysx
```

Once running, the sysx program prints the following banner and prompt:

```
System exerciser control program
Type h for help

>
```

The > is the command-mode prompt and indicates that the sysx program is ready to accept commands. Essentially, the sysx program has two modes of operation: command mode and run mode. To indicate modes, it issues a > prompt during command mode and a run> prompt during run mode.

Commands available in command mode are:

<CTRL/D>	Exit from the sysx program.
<CTRL/C>	Cancel current command and return to the prompt.
! command	Execute an ULTRIX-11 command.
b	Backup, save an existing log file.
c	Create an exerciser run script.
d	Delete an exerciser run script.
l	Print log files on the terminal or line printer.
n	Name the exerciser to run on each device.
p	Print the contents of a script.
r	Run an exerciser script.
s	Stop all exercisers.
x	Print a list of the exerciser run scripts.

Commands available in run mode are:

<CTRL/D>	Exit from the sysx program.
<CTRL/C>	Cancel current command and return to the prompt.
! command	Execute an ULTRIX-11 command.
l	Print log files on the terminal or line printer.
p	Print the status of the currently running script.
r	Restart system exercisers.
s	Stop system exercisers.

For further information, either use the sysx help command or read Appendix B, Sysx Program Example.

The next four sections discuss creating an exerciser run script, running an exerciser script, monitoring operations, and stopping an exerciser script.

NOTE

If you are running exerciser modules without the aid of the sysx program, you can stop them in one of two ways. First, you must be the superuser (root login) and must be in the /sysxr directory. Then, you can use either the sysxstop command alone or the sysx program and specify the s command. In either case, the system stops all running exerciser modules, even those that were invoked by other users. In addition, you should redirect all exerciser output to separate log files. Otherwise, if you are running more than one module, the output may become intermixed and, therefore, unreadable.

7.1.1 Creating Exerciser Run Scripts

To tailor a run script to your system configuration, use the sysx program and specify the c command. Each exerciser run script is a shell command file that contains the required commands to start the exerciser modules for:

- A single device
- A group of devices
- A CPU cluster (processor, memory, and floating point)
- The entire system

Each line in this script contains:

- Exerciser module name
- Options list
- Log file name

To obtain a list of currently existing run scripts, use the sysx x command.

To create a run script, use the sysx program and specify the c command. For example, the following sysx session creates an exerciser run script with the name "test".

```
System exerciser control program
Type h for help


> c

Script name <sysxr> ? test

Script exists, overwrite it <no> ? y

To cancel a script entry, type <CTRL/D> !
Answer any question with a '?' for help !

Exerciser name ?
```

Once running, the c command prompts first for the script name and then for the exerciser name. The default answer for each prompt is enclosed in angle brackets < >. To use a given default answer, press the <RETURN> key. For on-line help, type ? and press the <RETURN> key. To cancel a current entry and return to the module name prompt, press <CTRL/D>. To stop the c command, press the <RETURN> key in response to a module name prompt.

For further information, read Appendix B, Sysx Program Example.

When creating an exerciser run script, consider:

- Although you should run one disk exerciser for each drive, you should not run a disk exerciser on the system disk unless it is the only drive on the system.

- The magnetic tape exerciser handles various types of devices or drives. Because the tape modules exercise all drives on a controller, you should run a tape exerciser for each controller that is present on the system.

- The communications exerciser handles various types of communications devices. You should run a communications exerciser for each communications device that is present on the system. For the DL11 communications device, however, you should run one copy of the communications device exerciser for every 16 units. For example, when two copies are run, the first exercises the first 16 units, while the second exercises the second 16 units.

- To exercise all of memory, you should run only one copy of the memory exerciser.

- DIGITAL recommends that you run only two floating point

exerciser modules at the same time. Running more than two floating point exercisers simultaneously loads down the CPU and does not actually work the floating point unit as hard as the recommended two copies.

- You can run up to 50 copies of the CPU exerciser simultaneously. If you exercise only the CPU, you can run the maximum number of copies allowed. If you exercise only the CPU cluster, DIGITAL recommends that you run no more than six copies. If you exercise the entire system, however, DIGITAL recommends that you run only one CPU exercise module.

- To exercise the line printer, you should run the line printer exerciser.

7.1.2 Running Exerciser Scripts

To obtain information about how to run exerciser scripts, use the sysx program and specify the help command. In response to the > prompt, type:

```
> h r
```

If you are exercising disk or tape units, you should mount the appropriate scratch media before starting an exerciser run script. Then, to start an exerciser script, specify the r command. In response to the > prompt, type:

```
> r
```

Once running, the r command first prompts for the script name. To use the default script name (sysxr), press the <RETURN> key . To use a different script, type the appropriate name and press the <RETURN> key.

During the time that the sysx program is waiting for start-up confirmation, you may cancel the wait loop by pressing <CTRL/C>. When all modules are running, the sysx program returns this prompt:

```
run>
```

As it starts each exerciser module, the sysx program prints the module name and the start date and time. If a module fails to start, it also prints an error message. For further information about a reported start failure, examine the log file.

NOTE

When using the sysx program to control exerciser run scripts, you also can use the r command to restart modules that previously had been stopped. For further information, either use the help command to obtain information about the r command or read Appendix B, Sysx Program Example.

The following is a sample sysx session that uses a default run script:

```
# sysx


System exerciser control program
Type h for help


> r

Script name <sysxr> ?

Disconnect any customer equipment that may be affected
by test data transmitted on DH, DHU, DHV,
DZ, DZV, DZQ, DL output lines !

Confirm <no> ? y

Log files will be overwritten !
The `b' command may be used to save log files.

Proceed <no> ? y

Waiting for startup confirmation from exercisers.
Typing <CTRL/C> will cancel wait loop !

  cpx_01 started - Fri Jul 27 14:05:23 1984

 cmx_dh2 started - Fri Jul 27 14:05:30 1984


run>
```

7.1.3 Monitoring Operations

When you specify that exerciser messages and output are to be redirected to log files, the sysx program monitors each log file and periodically reports any size changes.

Normally, a change in the size of a log file indicates that an error has occurred.

The sysx program also provides two commands that let you monitor the script run directly. Once a script is running, specify the p command to print each module name and its corresponding run status. In response to the run> prompt, type:

```
run> p
```

Once a script is running, specify the l command to print the contents of one or more log files either at your terminal or at the line printer. In response to the run> prompt, type:

```
run> l
```

7.1.4 Stopping Exerciser Scripts

To stop one module, all copies of a module, or the entire exerciser run script, use the sysx program and specify the s command. In response to the run> prompt, type:

```
run> s
```

If you stop a specific module, you can restart it without affecting any other exercisers that were started by that script. During the time that the sysx program waits for an exerciser to stop, you may interrupt this wait loop by pressing <CTRL/C>. If you stop all the exercisers in the run script, the sysx program prints a command-mode prompt:

```
>
```

As it stops each exerciser module, the sysx program prints the module name and the stop date and time. If a module fails to stop, it also prints an error message. For further information about a reported stop failure, examine the log file.

To exit the sysx program once all modules have stopped and return to the ULTRIX-11 command interpreter, press <CTRL/D>.

NOTE

Normally, it can take several minutes for every exerciser to stop. But the RX02 exerciser can take even longer, because of the noninterruptible diskette format operation which occurs at the beginning of each pass.

7.2 Interpreting the Results of an Exerciser Run

The system places the results of the exerciser script run in both the exerciser log files and the system error log. Specifically, the exerciser modules report unrecoverable errors to the log files, while the ULTRIX-11 operating system logs recoverable errors in the system error log file.

Each exerciser log file contains this information about each exerciser run:

- Date and time started
- Date and time stopped
- End-of-Pass messages
- Periodic I/O statistics

When you stop an exerciser, the system also writes an error log summary report into the exerciser log file. This report gives a summary of the errors logged while that exerciser was running. To obtain detailed information about individual errors, use the ULTRIX-11 error logger and specify the full error report command. For further information, read Chapter 8, ULTRIX-11 Error Logger.

7.3 Exerciser Modules

When you use the sysx program to create a run script, you enter the required information to the appropriate prompt. Then, the sysx program automatically formats this information and creates the exerciser run script. You neither need to specify nor even need to know the exact command syntax. The information presented in the remainder of this chapter is for reference only.

The next seven sections describe an exerciser module:

- Options used
- Tests performed
- Messages displayed

Many of the exerciser modules share options. Some of the options, however, affect these modules differently. For ease of presentation, Section 7.4, Exerciser Options, discusses the format of each option and its applicable function for each module.

7.3.1 Communications Device Exerciser

The communication device module (cmx) has the -b, -d, -e, -h, -i, -l, -m, -n, and -u options. For further information, read Section 7.4, Exerciser Options.

This module exercises one or more lines on the DH11/DHU11/DHV11, DZ11/DZV11/DZQ11, and DL11 communication devices. This module treats the first 16 DL11 devices as unit 0 and the next 16 DL11 devices as unit 1. It, however, never exercises DL11 unit 0, line 0 (system console). The communications device module exercises a device in either maintenance loopback mode or line turnaround mode.

In maintenance loopback mode, the cmx module exercises each device by looping transmitter output back to receiver input and by transmitting data message packets on the selected lines. It compares the received message packet with the transmitted packet.

For the DH11 and DZ11/DZV11/DZQ11 devices, maintenance loopback mode loops back all lines on the multiplexer. This requires that no lines on the multiplexer be in use. The DHU11/DHV11 and DL11 communications controllers allow maintenance loopback on individual lines. This allows the use of maintenance loopback mode to test lines while other lines on the mux are used.

In line turnaround mode, place a line turnaround connector on the lines to be tested and specify the -m option. Then, the cmx module exercises the device by comparing the received data with the transmitted data. This mode is

useful for testing individual lines on DH11 and DZ11/DZV11/DZQ11 multiplexers. You also can exercise the second SLU supplied with the PDP-11/24 processor as a DL11.

You can set the transmission bit rate on the selected lines to either a fixed rate or a randomly varying bit rate.

The transmitted data message packet consists of:

- Device unit number (byte)
- Line number
- Data characters (incrementing pattern and random number)
- Checksum packet

The transmission sequence consists of:

1. The exerciser transmits the data message packet on the selected line.

2. The selected line receives the packet.

3. The exerciser uses the checksum and character counts to validate the received packet.

4. The exerciser checks the unit and line numbers to ensure the selected line received the packet.

5. The exerciser compares the data portion of the received packet with the data in the transmitted packet.

The error messages for the communications device module have this format:

```
Read data timeout on DH11 unit 0 line 14
Complete data packet not received within 2 minutes

************ READ DATA ERROR *************
         Fri Aug 20 23:05:24 1982

Data transmitted on DH11 unit   0 line  14
Data received    on DH11 unit   0 line  14
Data transmitted at 9600 bits/second
Data packet checksum was BAD
129 characters transmitted
125 characters received

CHAR #  GOOD    BAD

    99  012     016
   100  013     017
   101  014     020
   102  015     021
   103  016     022

[ data error print limit exceeded ]

*****************************************
```

The first two lines are the read data timeout message. This message indicates that the selected line did not receive the data message packet within two minutes after the start of transmission. Printing only the timeout message indicates that the selected line received none of the message packet.

The data mismatch error message may follow the timeout message. This indicates that the selected line received an incomplete message packet. Printing only this message indicates that the selected line received the entire message packet, but either the data does not match or the checksum is incorrect. The data mismatch error message provides information about the message packet and prints the actual mismatched data in the packet.

NOTE

In the example above, the printed mismatched data was limited to five lines by using the -n option.

7.3.2 CPU Exerciser

The CPU module (cpx) has no options.

This module runs a compute-bound process that exercises the processor by repeatedly reproducing the activity normally associated with creating and executing user processes in the ULTRIX-11 operating system environment. The CPU exerciser test four main areas:

- Thirteen C programming functions
- Dynamic memory reallocation
- Dynamic growth of the user stack
- The fork() and exec() system calls

The dynamic memory test uses the calloc() function to obtain additional memory from the operating system. The user stack test expands the stack size beyond the twenty 64-byte segments initially allocated by the system. The fork and exec test relocates the cpx process throughout memory.

Because it is unlikely to encounter problems while performing any of these tests, the CPU module does minimal error checking and generates cryptic error messages. If one of these tests does fail, the ULTRIX-11 operating system probably will crash.

The error messages for the CPU exerciser have this format:

```
****** CPU EXERCISER ERROR ******
****** TEST ## SUBTEST ### ******
```

The test numbers and functions are:

TEST	FUNCTION
1	C language - if and goto statements
2	C language - while, break, and continue statements
3	C language - do-while, break, continue statements
4	C language - for, break, and continue statements
5	C language - switch and break statements
6	C language - shift operators
7	C language - relational/equality operators
8	C language - bitwise AND, OR, XOR operators
9	C language - logical AND, OR conditional operators
10	C language - assignment operators
11	C language - function calling and argument passing
12	C language - function argument return
13	C language - string move function
14	Crunch integers in data space
15	Sort integers in data space (uses calloc() function)
16	Crunch data structures on the user stack and force dynamic stack growth
17	Crunch a union on the user stack

7.3.3 Disk Exercisers

There are six disk exerciser modules:

Module	Disks
hpx	RM02/3/5, RP04/5/6, ML11
hkx	RK06/7
hxx	RX02
rpx	RP02/3
rlx	RL01/2
rkx	RK05
rax	RA60/RA80/RA81, RX50/RD51/RD52, RC25

The disk exercisers differ only where required by the characteristics of the disk being exercised. You can use the hpx module with any combination of the listed disks on the first, second, or third RH11/RH70 MASSBUS disk controller. You can mix the ML11 solid state disk with the RM and/or RP disks either on the same or on a separate RH controller.

The disk exerciser modules have these options:

hpx	hkx	rpx	rlx	rkx	rax	hxx
-c						
-d	-d	-d	-d	-d	-d	-d
-e	-e	-e	-e	-e	-e	-e
-f	-f	-f	-f		-f	
-h	-h	-h	-h	-h	-h	-h
-i	-i	-i	-i	-i	-i	-i
-m	-m	-m			-m	-m
-n	-n	-n	-n	-n	-n	-n
-s	-s	-s	-s	-s	-s	-s
					-w	
					-x	

For further information, read Section 7.4, Exerciser Options.

Each module begins by printing the selected disk drive's status (on-line or off-line). The module also prints a list of any read-only file systems. For further information about file systems and disk partitioning, read Section 1.2.2, File System, and Section 1.3, Logical Partitioning of Disks.

The module treats a file system as read-only if it:

- Is mounted
- Contains root file system (system disk)
- Contains the error log
- Contains the swap area
- Overlaps any of the above areas

A mounted file system is one that the system is currently using. If a disk that contains either user files or other

volatile data is on-line, ready, but not mounted, the disk exercisers write on that disk and destroy its data. Therefore, you should remove all disk packs that contain valuable data before running any disk exercisers. The rax module protects customer data on fixed-media disks by allowing writes to the maintenance area only.

The hxx exerciser (RX02) formats the diskette for single (RX01) or double (RX02) density at the beginning of each pass.

The ULTRIX-11 device drivers for the partitioned disks (except RL01/2) contain a size table which specifies the sizes and locations of the logical file systems on the disk. The exercisers for these disks (hkx, hpx, rax, and rpx) contain a copy of the disk driver size table to ensure they do not write on any of the read-only file systems.

In the unlikely event that the size table in the exerciser does not match the size table in the disk driver, the exerciser prints this error message:

```
hpx: unit 2 sizes mismatch !
```

The disk exercisers write test data patterns to the disk at random addresses, read the data back, and compare the data in the write buffer with the data in the read buffer.

The disk exercisers attempt to simulate ULTRIX-11 file I/O by using random length transfers in block I/O mode. They also simulate swapping and other physical I/O by doing long transfers in raw I/O mode.

The basic test sequence consists of:

1. For large disks only, the exerciser randomly selects the file system and starting block number of the transfer. The exerciser also selects the transfer size (512 bytes, a multiple of 512 bytes, or a random number of bytes).

2. For all disks, the exerciser selects the test data pattern to be used for the transfer. This is either the worst-case data pattern or a random data pattern.

3. For all disks, the exerciser loads the test data into the write buffer. Then, starting at the specified block number, it writes this data out to the disk.

4. For all disks, the exerciser reads from disk a randomly selected block, one that was not part of the previous transfer. The exerciser does not test the data read from this block.

5. For all disks, the exerciser first clears the read

buffer and then reads in the data that was previously written out to disk.

6. For all disks, the exerciser compares the data in the read buffer to the data in the write buffer and reports any mismatches.

Each disk exerciser repeats this sequence until you stop it manually.

The error messages for the disk exercisers have this format:

```
******
HARD DISK ERROR - Sat Sep 11 10:20:34 1982
Returned byte count = -1 (-1 = error)
Error type: I/O error

unit  filesys  block    xfer size  xfer type
1     7        43836    3072 bytes  RAW I/O WRITE
******


******
HARD DISK ERROR - Sat Sep 11 10:20:35 1982
Returned byte count = -1 (-1 = error)
Error type: I/O error

unit  filesys  block    xfer size  xfer type
1     7        43836    3072 bytes  RAW I/O READ


Write was from word 1848 of write buffer
Read  was to   word 6526 of read  buffer

DATA COMPARE ERROR - BLOCK 43837

Write buffer address = 2104
Read  buffer address = 6782

WORD = 0
GOOD = 165555
BAD  = 000000

WORD = 1
GOOD = 133333
BAD  = 000000

WORD = 2
GOOD = 165555
BAD  = 000000

WORD = 3
GOOD = 133333
BAD  = 000000

WORD = 4
GOOD = 165555
BAD  = 000000

[error printout limit exceeded]
******
```

In this example, the message shows an unrecoverable write error on block 43836 followed by a fatal read error on the same block. The following information also is printed:

byte count — Indicates the actual number of bytes transferred (-1 indicates a fatal error).

Error type — Indicates the error type returned to the exerciser. For further information about error codes, read intro(2) in the ULTRIX-11 Programmer's Manual, Volume 1.

unit — Indicates the physical unit number of the failing disk drive.

filesys — Indicates the logical subunit of the disk drive where the I/O operation occurred. (This number is always 0 for the nonpartitioned RK05 disks.) For the hxx exerciser (RX02), the word "density" replaces "filesys" and indicates either single or double density access mode.

block — Indicates a logical block number relative to the start of the logical subunit specified by the filesys column. This is not the start of the disk. For the logical partition layout of each disk, read Appendix D, Disk Logical Partition Sizes.

xfer size — Indicates the size of the transfer in bytes.

xfer type — Indicates the type of I/O operation (read/write and block/raw I/O mode).

The write and read buffer addresses are relative to the start of the buffers and are not physical memory addresses.

In the above example, the amount of information was severely limited because the ULTRIX-11 operating system did not pass any error information back to the disk exerciser. The system only informs the exerciser of the occurrence of hard (unrecoverable) errors. It does not report soft (recoverable) errors to the disk exercisers.

The ULTRIX-11 operating system saves detailed error information for hard and soft errors in the system error log file. To obtain a detailed report about the disk errors, use the error log print command, elp. Stopping the disk exerciser produces an error log summary report. The report summarizes all the errors occurring on the disk being exercised.

7.3.4 Floating Point Exerciser

The floating point exerciser (fpx) has the -e and -n options. For further information, read Section 7.4,

Exerciser Options.

This module exercises either the FP11-type floating hardware or the kernel-resident floating point simulator by testing various arrays of floating point numbers. If FP11 hardware is not on the processor or if the kernel-resident floating point simulator either is not configured or is turned off, the module simply exits. The floating point module also exercises the exception mechanism by running two copies of itself to produce floating point exceptions (traps through location 244). Because the ULTRIX-11 system does not support the PDP-11/40 Floating Instruction Set (FIS), you cannot use the floating point exerciser on a PDP-11/40 with the FIS option.

The floating point tests are:

TEST 1 - Increment/Decrement

Starting at 0.0 and incrementing by 0.1, the exerciser increments two floating point numbers 99,999 times. After each increment, the exerciser compares the two numbers. Then, starting with the resulting numbers and decrementing by 0.1, the exerciser decrements the two floating point numbers 99,999 times. After each decrement, the exerciser compares the numbers. Finally, after completing this operation, the exerciser tests both numbers for 0.0 values.

TEST 2 - Floating Point Exceptions

The exerciser produces a floating-divide-by-zero exception by dividing 1.0 by 0.0. Using the ULTRIX-11 floating point exception mechanism, the exerciser verifies that the FP11 hardware status registers contain the correct values for the floating point divide by zero exception.

TEST 3 - Sort Floating Numbers

The exerciser copies an array of 16 presorted floating numbers to a scratch array in random order. The exerciser sorts these numbers and compares the result to the original presorted array.

TEST 4 - Floating Point Addition

The exerciser first adds two arrays of 16 floating point numbers. Then, it compares the result with a third array that contains the expected answers.

TEST 5 - Floating Point Subtraction

The exerciser first subtracts two arrays of 16 floating

point numbers. Then, it compares the results with a third array that contains the expected results.

TEST 6 - Floating Point Multiplication

The exerciser first multiplies two arrays of 16 floating point numbers. Then, it compares the result with a third array that contains the expected answers.

TEST 7 - Floating Point Division

The exerciser first divides two arrays of 16 floating point numbers. Then, it compares the results with a third array that contains the expected answers.

The error messages for the floating point exerciser have this format:

```
****** FPP EXERCISER ERROR ******
****** TEST NUMBER ##      ******
```

This message banner is followed by descriptive text that varies according to error type. Normally, the text describes the expected data, actual data, and information concerning the operation attempted.

7.3.5 Line Printer Exerciser

The line printer module (lpx) has the -h and -p options. For further information, read Section 7.4, Exerciser Options.

This module exercises the LP11 line printer controller and the line printer. It prints one test pattern that alternates ones and zeros and another pattern that increments through every printing character. The line printer exerciser prints 12 pages of test patterns and then pauses for 15 minutes.

When the system is using the line printer, the line printer module prints this message:

```
lpx: Printer in use by spooler,
     waiting for print job to finish !
```

The line printer module then waits for the system to release the printer. When this occurs, it begins printing the test patterns. When it cannot open the line printer module, the line printer module prints this message:

```
lpx: Can't open LP, check for LP off-line.
     Will retry at one minute intervals.
```

If the printer is off-line, you should place it back on-

line. The line printer module automatically retries at 1-minute intervals.

7.3.6 Memory Exerciser

The memory module (memx) has no options.

This module exercises all memory, except that occupied by the ULTRIX-11 operating system, by dividing memory into several equal sections. The actual number depends on the amount of available memory, but the normal range is between 5 and 50. Then, it creates a subprocess (copy of itself) for each remaining section of memory.

The master memx exerciser starts one of the subprocesses, waits for it to complete exercising its section of memory, and then starts the next subprocess. When all of the subprocesses are completed, the exerciser creates a new set of subprocesses. The exerciser repeats this procedure until you stop it manually.

In byte-access mode, each subprocess exercises its section of memory with a checkerboard pattern and its complement. In word-access mode, each subprocess exercises its section of memory with a walking one pattern and, its complement, a walking zero pattern.

When you use the memory module in conjunction with other exerciser modules, it forces swapping activity. Swapping occurs because the exerciser attempts to use all the memory on the system, some of which is occupied by other exercisers. When the ULTRIX-11 operating system has more processes to run than free memory in which to run them, it must swap out processes.

If your system configuration has nonparity memory, or if a data error without a memory parity error occurs, the memx error messages are in the format:

```
Memory data error [byte access]
Memory exerciser process I.D. = #
Virtual address = ######
Expected data   = ######
Actual data     = ######
```

The data error indicates the memory access mode (word or byte). The process ID indicates the subprocess that experienced the error. The address indicates where the data mismatch occurred. The expected and actual data also are indicated. Because the subprocess has a short lifespan, determining the physical address is very difficult.

When your system configuration has parity memory, any memory data error is usually accompanied by a memory parity error

(trap through location 114). When a memory parity error occurs in the section of memory that a subprocess is exercising, the system immediately suspends running that subprocess. This occurs before the system can print the data mismatch error message cited above. For further information about parity memory errors, read Section 8.4.3, Memory Parity Record, and Section 10.2.15, panic: parity.

When a memory parity error occurs, the master memx exerciser is notified of the subprocess's abnormal termination. In turn, it also prints this message:

Memory exerciser (pid # status ###) terminated abnormally !
Check error log for error at about - Sat Mar 28 11:10:10 1982

The pid number indicates the process ID of the subprocess that terminated abnormally. The status indicates the reason (exit status) for termination. For further information about exit codes, read signal(2) and wait(2) in the ULTRIX-11 Programmer's Manual, Volume 1.

The date and time indicated is the most important information in this message. When a memory parity error terminates a subprocess, the system makes an entry in the error log that describes the memory parity error.

When a subprocess does not complete exercising its section of memory within the 15-minute time limit, the master memx exerciser terminates the subprocess, prints this message, and continues exercising memory:

memx: memxr (pid #) timed out !

The # indicates the process identification number of the subprocess.

7.3.7 Tape Exercisers

The magnetic tape module (mtx) has the -d, -e, -f, -h, -i, -n, and -s options. For further information, read Section 7.4, Exerciser Options. In addition, this module exercises all available tape units connected to any of these controllers:

Mnemonic	Controller/Unit
ht	TM02/3 with TU16/TE16/TU77
tm	TM11 with TU10/TE10/TS03
ts	TS11/TSV05/TU80/TK25

This module first writes a number of records that contain test data patterns onto the tape. Then, it reads each record and matches the data against the test pattern.

The mtx exerciser performs the following tests at either 800 or 1600 bpi and in either block or character (raw) I/O mode:

TEST 1 - Short File Test

The short file test writes a number of 512-byte records from the write buffer to the tape, reads the same number of records back from the tape into the read buffer, and matches the data. The number of records starts at 1, increments to 16, and then decrements back to 1. Then, it verifies that the correct number of records were written to tape. It attempts to read one extra record and checks for an end-of-file error. The write and read buffer addresses are rotated throughout the exerciser's in-memory data buffer.

TEST 2 - Variable Length Record Test

The variable length record test writes a single tape record that starts at 512 bytes and increments by 512 bytes to a maximum 10240 bytes. The exerciser reads the tape record and matches the data against the test data pattern.

TEST 3 - Large File Test

The large file test simulates very large files by writing enough records to fill the length of tape specified by the -f option. The default length is 500 feet. Then, it verifies that the correct number of records were written to tape. It attempts to read one extra record and checks for an end-of-file error. The tape record size is 512 bytes in block I/O mode and 10240 bytes in raw I/O mode. For the 512-byte records only, the exerciser rotates the write/read buffer addresses through its in-memory buffer area.

Depending on the condition, the error messages for the magnetic tape module have differing formats. When the module could not open the tape drive, it prints these messages:

```
mtx: can't open /dev/mt0 [off-line]

mtx: [will retry for at least 15 minutes, then quit !]
```

The drive may be off-line, write locked, or already open. The module periodically attempts to reopen it for 15 minutes. If an attempt is successful, it continues exercising the tape. If the attempts still fail after the 15-minute time limit, the exerciser prints this message and then terminates:

```
mtx: FATAL TAPE ERROR - can't open /dev/mt0 after 15 minutes
```

When read or write errors occur, the exerciser prints this message:

```
******
TEST 3 - HARD TAPE READ ERROR - Fri Sep 10 16:08:42 1982
Returned byte count = -1 (-1 = error)
Error type: Fatal error - tape position lost

unit    density I/O     record  # of    record
number  BPI     mode    number  records length

0       1600    raw     327     513     10240 bytes

Write buffer address = 0
Read  buffer address = 0

DATA COMPARE ERROR - RECORD 327

WORD =   3584
GOOD = 137500
BAD  = 137400

WORD =   3585
GOOD = 137500
BAD  = 137400

WORD =   3586
GOOD = 137500
BAD  = 137400

WORD =   3587
GOOD = 137500
BAD  = 137400

WORD =   3588
GOOD = 137500
BAD  = 137400

[error printout limit exceeded]
******
```

In this example, the error occurred during TEST 3 (large file test) and was an unrecoverable read data error at 1600 bpi in raw I/O mode on unit 0. The error occurred on record number 327 of 513 records, and the record length was 10240 bytes. The write and read buffer addresses given are not physical memory addresses. They are the offset from the start of the buffer where the actual write or read operation begins.

The data mismatch printout shows the address where the mismatch occurred and is expressed as a word offset from the start of the transfer (buffer address). The message also

shows the good and bad data. Because the ULTRIX-11 operating system indicates only that a fatal error has occurred to the tape exerciser, it is not possible for the module to produce a detailed error printout. The ULTRIX-11 operating system does save the detailed error information in the error log file. To obtain a detailed error report, use the error log printout command, elp. When an error stops the exerciser, the module produces an error log summary report of all errors for the current module run.

Some of the tests verify that the system writes the correct number of records to tape. If these verification tests fail, the exerciser prints the same error message as in the previous example but omits the data mismatch printout. Then, it prints this message:

```
[missing EOF or extra record(s) at end of file]
```

7.4 Exerciser Options

Although many of the exerciser modules share options, the options may designate different functions for each. This section is in alphabetical order by option mnemonic. The accompanying text describes the format and effect that are applicable for each module.

7.4.1 -b Option

The -b option is used with the cmx communications device exerciser only. It specifies the speed (bps) for all lines on that device and has this format:

```
-b #
```

The # indicates the speed (bps). For example, the following command sequence indicates that cmx is to exercise all lines on DZ11, unit 0 at 9600 bps:

```
cmx -dz0 -b 9600
```

7.4.2 -c Option

The -c option is used with the hpx and rax disk exercisers. When used with either exerciser, it specifies the disk controller and has this format:

```
-c#
```

The hpx exerciser supports various combinations of RM02/3/5, RP04/5/6, and ML11 disks connected to as many as three RH11 or RH70 controllers. The ULTRIX-11 RH controller number is not specified by the physical or electrical position of the RH controller on the bus. The following numbers are used as controller indicators:

0 - First RH controller with RM02/3/5, RP04/5/6 and/or ML11 disks. The disks are referred to as "hp".

1 - Second RH controller with RM02/3/5, RP04/5/6 and/or ML11 disks. The disks are referred to as "hm".

2 - Third RH controller with RM02/3/5, RP04/5/6 and/or ML11 disks. The disks are referred to as "hj".

The rax exerciser supports the following MSCP controllers:

```
UDA50/UDA50A - for RA60/80/81 disks
RQDX1 - for RD51/RD52/RX50 disks
KLESI - for RC25 disks
RUX1 - for RX50 disks
```

Controllers are assigned numbers based on the order that they were specified during the system generation. To determine the controller number for the device you wish to exercise, use the sysx help information to obtain a list for your configuration. In addition, you also may use the rasize command. For further information, read rasize(1M) in the ULTRIX-11 Programmer's Manual, Volume 1.

7.4.3 -d Option

The -d option is used with the cmx communications device exerciser; the mtx magnetic tape exerciser; and the hkx, hpx, hxx, rax, rkx, rlx, and rpx disk exercisers.

When used with the cmx exerciser, it specifies the device type and unit number and has this format:

```
-d?#
```

The ? indicates the device type, and the # indicates the unit number.

When used with the mtx exerciser, it specifies the drive number and has this format:

```
-d#
```

For example, the following command sequence indicates that mtx is to exercise drives 0 and 1 on the TM02/3 controller:

```
mtx -ht -d0 -d1
```

If you are using the mtx exerciser and do not specify the -d option, the module exercises all available drives on the designated controller (default).

When used with the hkx, hpx, hxx, rax, rhx, rlx, and rpx exercisers, it specifies the desired disk unit number and has this format:

```
-d#
```

The # indicates the disk unit number.

7.4.4 -e Option

The -e option is used with the cmx communications device exerciser; the fpx floating point exerciser; the mtx magnetic tape exerciser; and the hkx, hpx, hxx, rax, rkx, rlx, and rpx disk exercisers.

When used with any of these exercisers, it specifies the number of errors that can occur on the designated device before the exerciser stops. This option is useful for

preventing a faulty device from producing an excessive number of errors and extremely large log files. It has this format:

```
-e #
```

The # indicates the error limit. If you use any of these exercisers and do not specify the -e option, the error limit is set to 100 (default).

7.4.5 -f Option

The -f option is used with the mtx magnetic tape exerciser; and the hkx, hpx, rax, rlx, and rpx disk exercisers.

When used with the mtx exerciser, it specifies the tape length and has this format:

```
-f#
```

The # indicates the length of the tape in feet. If you use the mtx exerciser and do not specify the -f option, the module uses 500 feet (default).

When used with the hkx, hpx, rax, rlx, and rpx exercisers, it specifies the disk partition (file system) that is to be exercised and has this format:

```
-f#
```

The # indicates the partition number (file system). If you use the hkx, hpx, rax, rlx, and rpx exercisers and do not specify the -f option, the module exercises all partitions (default). For further information about disk partitioning, read Section 1.3, Logical Partitioning of Disks.

7.4.6 -h Option

The -h option is used with the cmx communications device exerciser; the lpx line printer exerciser; and the hkx, hpx, hxx, rax, rkx, rlx, and rpx disk exercisers.

When used with any of these exercisers, it provides help text about the other exerciser options and has this format:

```
-h
```

For example, the following command sequence provides help text on the cmx option.

```
cmx -h
```

7.4.7 -i Option

The -i option is used with the cmx communications device exerciser; the mtx magnetic tape exerciser; and the hkx, hpx, hxx, rax, rkx, rlx, and rpx disk exercisers.

When used with any of these exercisers, it suppresses all start-up or status messages and has this format:

```
-i
```

Before starting data transmission on a line, the cmx exerciser prints a warning message, delays for one minute, and prompts for verification before starting. This ensures that there is no sensitive customer equipment connected to the line that you want to exercise.

Before starting, the mtx exerciser prints out the status of the designated drives.

Before starting, the disk exercisers print out the status of all read-only file systems on the designated drives. If any of the file systems contain data, the exercisers treat it as read only and do not overwrite data.

7.4.8 -l Option

The -l option is used with the cmx communications device exerciser only. It specifies the line that is to be exercised and has this format:

```
-l # #
```

The first # indicates the desired line, while the second # (optional) indicates speed (bps). In addition, multiple uses of the -l option can appear in one command sequence. For example, the following command sequence specifies a DH11, unit zero with line 1 (9600 bps) and line 2 (300 bps) exercised:

```
cmx -dh0 -l 1 9600 -l 2 300
```

If you use the cmx exerciser and do not specify the -l option, all lines not disabled with the -u option are exercised (default).

7.4.9 -m Option

The -m option is used with the cmx communications device exerciser; and the hkx, hpx, hxx, rax, and rpx disk exercisers.

When used with the cmx exerciser, it suppresses maintenance loopback mode on all devices and has this format:

```
-m
```

For further information about cmx test modes, read Section 7.3.1, Communications Device Exerciser.

When used with the hkx, hpx, rax and rpx disk exercisers, it simply lists the partition layout of the selected drive and has this format:

```
-m
```

Then, after displaying the partition layout, the exerciser exits.

When used with the hxx disk exerciser, it specifies the exercise mode that is to be used on the RX02 drive and has this format:

```
-m#
```

The # indicates the desired exercise mode. The RX02 disk can be exercised in either RX01-compatibility mode or in RX02 mode. To use RX01 mode, enter a 1 as the mode indicator (for example, -m1). To use RX02 mode, enter a 2 as the mode indicator (for example, -m2). If you use the hxx exerciser and do not specify the -m option, the RX02 drive is exercised in both modes (default).

7.4.10 -n Option

The -n option is used with the cmx communications device exerciser; the fpx floating point exerciser; the mtx magnetic tape exerciser; and the hkx, hpx, hxx, rax, rkx, rlx, and rpx disk exercisers.

When used with any of these exercisers, it specifies the number of data mismatch errors to print for each error occurrence. For example, a disk exerciser may generate as many as 256 data mismatch errors for a single disk sector transfer. This option lets you limit the output to a reasonable number and has this format:

```
-n#
```

The # indicates the desired number of data mismatches. By limiting the number of errors, you not only save space in the log file but also, when printing error reports, save paper.

7.4.11 -p Option

The -p option is used with the lpx line printer exerciser only. It specifies the pause time before resuming printing

and has this format:

```
-p#
```

The # indicates the desired number of minutes. To save paper, the lpx exerciser generates about 12 pages of actual printout and then goes into pause state. During this pause state, it continues to exercise the line printer controller by sending nonprinting characters (NULLS) to the printer. For continuous printing (no pause), enter 0 as the minute indicator. If you use the lpx exerciser and do not specify the -p option, the pause time is set to 15 minutes (default).

7.4.12 -s Option

The -s option is used with the cmx communications device exerciser; the mtx magnetic tape exerciser; and the hkx, hpx, hxx, rax, rkx, rlx, and rpx disk exercisers.

When used with any of these exercisers, it specifies that I/O statistics are to be printed at the designated time interval and has this format:

```
-s#
```

The # indicates the desired time interval in minutes. If you use any of these exercisers and do not specify the -s option, I/O statistics are printed out every 30 minutes (default).

7.4.13 -u Option

The -u option is used with the cmx communications device exerciser only. It specifies that the designated line is to be omitted and has this format:

```
-u #
```

The # indicates the designated line. For example, the following command sequence specifies that cmx is to exercise all lines on the DH11, unit zero, except line 1:

```
cmx -dh0 -u 1
```

7.4.14 -w Option

The -w option is used with the rax disk exerciser only. It specifies that the exerciser is to use the entire disk but with the normal write protections (no writes to root, swap, and mounted file systems or to error log). It has this format:

-w

The RA80, RA81, RD51, and RD52 disks contain fixed Winchester disks. The RC25 contains both a fixed and removable Winchester disk. The RA60 is a removable disk that operates on the same controller as the RA80 and RA81. To protect data, the ULTRIX-11 operating system reserves a small maintenance area at the end of the disk as a free-fire zone. This maintenance area lets the rax exerciser have write and read access to the disk without overwriting data in the customer area. If you use the rax exerciser and do not specify the -w option, the exerciser reads the entire disk but performs write operations to the maintenance area only (default).

7.4.15 -x Option

The -x option is used with the rax disk exerciser only. It specifies that the drive to be tested is an RX50 and that writing to any area of the disk is allowed as long as the normal file system protection mechanisms are satisfied. It has this format:

-x

Because it does not have a maintenance area, the RX50 diskette drive differs from the other drives exercised by the rax exerciser. When this option is specified, the exerciser first verifies that the drive to be tested is indeed an RX50. If the drive is not an RX50, the exerciser stops. This prevents accidentally writing to a nonmaintenance area on drives other than an RX50. When you use the sysx program to generate a run script, it automatically specifies the -x option when the designated drive is an RX50.

7.5 End-of-Pass Messages

Every exerciser, except the communications exerciser (cmx), prints periodic end-of-pass messages. These messages give a general indication of exerciser progress.

The amount of time an exerciser requires to make a pass varies depending on the type of processor and the number of exercisers running. For example, when running alone, the CPU exerciser makes a pass in less than five minutes. But, when you are exercising the whole system, it can take over 30 minutes.

Chapter 8

ULTRIX-11 Error Logger

The ULTRIX-11 kernel monitors system performance and records error data in an in-memory error message buffer. In addition, the ULTRIX-11 error logger collects information about system and device errors as they occur and stores this information in the system error log file. The ULTRIX-11 error logger consists of:

elc Runs in background and periodically copies error data from the in-memory message buffer to the system error log file.

eli Enables or disables error logging, saves the contents of the error log file, initializes the error log file, and prints the sizes of the error log file.

elp Formats and prints error reports.

The remaining sections of this chapter discuss:

- Operating procedures
- Error messages
- Report formats
- Sample reports

8.1 Operating Procedures

Normally, the ULTRIX-11 error logger operates automatically. The multiuser start-up and restart files, /etc/rc and /opr/restart, automatically enable system error logging and print the sizes of the error log file. In addition, during multiuser shutdown, the opser shutdown command automatically disables system error logging. During most daily operations, you will not need to attend to the ULTRIX-11 error logger.

When the error log file requires maintenance, however, you may have to operate the ULTRIX-11 error logger manually. The next four sections discuss the procedures for using the eli program to:

- Disable and enable error logging
- Save the contents of the error log file
- Initialize the error log file
- Print the size of the error log file

8.1.1 Disabling and Enabling Error Logging

On occasion, you may have to disable or enable error logging manually. For example, you may have to disable error logging to perform routine maintenance on the error log file. Or, you may want to enable error logging while the system is in single-user mode.

To disable error logging manually, use the eli program and specify the -d option. In response to the superuser prompt, type this command sequence:

```
# /etc/eli -d
```

Once error logging is disabled, you then can proceed with the planned maintenance.

To enable error logging manually, use the eli program and specify the -e option. In response to the superuser prompt, type this command sequence:

```
# /etc/eli -e
```

If error logging already is enabled, the system simply ignores this command. Once enabled, the error logger writes data to the error log file.

8.1.2 Saving the Error Log Contents

When maintenance requires you to initialize the error log file, you should save the current data before doing so.

To copy the contents of the error log file, use the eli

program and specify the -c option. In response to the superuser prompt, type this command sequence:

```
# /etc/eli -c file
```

file Specifies the file to which the error log data is to be copied.

Once the data contained in the error log file has been saved, you then can initialize the file.

8.1.3 Initializing the Error Log File

When the error log file is full and requires your attention, you should first save the contents of and then initialize the error log file.

To initialize the error log file, use the eli program and specify the -i option. In response to the superuser prompt, type this command sequence:

```
# /etc/eli -i
```

The eli program then prompts for confirmation. When you respond yes, the eli program zeroes out the error log file and initializes all the system's error logging pointers and buffers.

Although you can initialize the log file at any time, the system automatically disables error logging when you initialize the error log file when in multiuser mode. If you do so, therefore, you later must enable error logging manually. DIGITAL recommends that you initialize the error log file only when in single-user mode.

8.1.4 Printing the Sizes of the Error Log File

Although the system automatically prints the sizes of the error log file during a multiuser startup or restart, you also can print them at any other time.

To print the current size (bytes) and block usage of the error log file, use the eli program and specify the -u option. In response to the superuser prompt, type this command sequence:

```
# /etc/eli -u
```

The eli program then prints the current sizes of the error log file. For further information, read Section 8.2.1, Error Log File -- Blocks Used.

8.2 Error Messages

This section discusses those messages that may occur when running the ULTRIX-11 error logger.

8.2.1 ERROR LOG has - # of # blocks used

The eli program prints this message in response to queries about the size of the error log file (Section 8.1.4). Each time the system makes the transition from single-user mode to multiuser mode, the eli program prints this message. When the number of blocks used increases to within three blocks of the maximum error log file size, the eli program also prints this message each time an error is logged. This repetition warns that the error log file is nearly full and corrective action is needed. For further information, read Section 8.1.2, Saving the Error Log Contents, and Section 8.1.3, Initializing the Error Log File.

8.2.2 logger: MISSED ERROR

This message indicates that the in-memory error message buffer overflowed. As a result, one or more error messages may have been lost. After printing this message, the system disables error logging. Usually, the system prints this message when it writes data to the in-memory error message buffer faster than the elc program can remove and write it out to the error log file. This usually occurs when the elc program stops running for some reason.

8.2.3 logger: ERROR LOG DEVICE

This message indicates that a disk error occurred on a block within the error log file. After printing this message, the system disables error logging.

8.2.4 elc: bad error record, error logging disabled

This message indicates that the format of an error record was invalid. Normally, the elc program checks the format of each error record before it copies it from the in-memory buffer to the error log file. After printing this message, the system disables error logging. This can occur if the in-memory buffer is corrupted.

8.2.5 elc: error log full, logging disabled

When the error log file is full, the elc program can no longer copy data from the in-memory buffer to the error log file. When this occurs, the elc program prints this message, and the system then disables error logging. This message indicates that the error log file requires immediate attention. For further information, read Section 8.1.2, Saving the Error Log Contents, and Section 8.1.3,

Initializing the Error Log File.

8.2.6 elc: read error bn =

This message indicates that an error occurred as the elc program attempted to read from the error log file. The # specifies the block number relative to the start of the error log file. After printing this message, the system disables error logging.

8.2.7 elc: write error bn =

This message indicates that an error occurred as the elc program attempted to write to the error log file. The # specifies the block number relative to the start of the error log file. After printing this message, the system disables error logging.

8.3 Printing Error Log Reports

To produce readable reports from the data in the error log file, use the elp program. You can use this program to produce either summary or full reports. A summary report lists the number and types of errors for the system as well as for each device. In addition to this information, a full report provides a detailed description of each error. The elp program also has several options which can be used to limit the error reports to only the information desired. Many of these options can be used in combination.

To obtain on-line help information, use the elp program and specify the -h option. In response to the shell prompt, type this command sequence:

```
$ elp -h
```

The elp program then prints the following help information:

(elp) - ULTRIX-11 error log report generator.

This command formats and prints error reports based on the error data captured by the error logger.

Usage:

```
elp [-h] [-s] [-f] [-b] [-r] [-u] [-d sd ed] [-et[#]] [file]
```

elp — With no arguments, a summary of all errors is printed followed by a detailed report on each error.

-h — Print this help message.

-s — Print only the error summary report.

-f — Print only the detailed error report.

-b — Print only brief descriptions of each error.

-r — For block device errors, print only recovered errors.

-u — For block device errors, print only unrecovered errors.

-d — Print only the errors within the specified date/time range.

(sd) - Starting date/time (yymmddhhmmss).
(ed) - Ending date/time (yymmddhhmmss).

note - All digits must be present in date/time.

-et[#] — Print only the error for the specified error type. Only one error type may be specified. `#' - optional unit number for block device errors only.

file — Take the input from the specified file instead of the current error log.

The next five sections discuss:

- Summary and full error reports
- Error reports from saved error log contents
- Error reports by error type
- Error reports for hard and soft errors
- Error reports by date and time

8.3.1 Summary and Full Error Reports

When you do not specify options, the elp program first prints a summary report of all error data and then prints a full report that describes each error.

To print a summary report only, use the elp program and specify the -s option. In response to a shell prompt, type this command sequence:

```
$ elp -s
```

To print a full report only, use the elp program and specify the -f option. In response to a shell prompt, type this command sequence:

```
$ elp -f
```

In addition, to print a full report but limit the description of each error to a brief statement only, use the elp program and specify the -b option. In response to a shell prompt, type this command sequence:

```
$ elp -b
```

8.3.2 Error Reports from a Saved Error Log File

The elp program normally produces error reports with the data from the error log file. To produce either a summary or full report from data previously saved using the eli program, use the elp program and specify the appropriate option and alternate file name. For example, this command sequence produces a summary report from the data contained in the olderrors file:

```
$ elp -s olderrors
```

To save the contents of the current error log file, use the eli program. For further information, read Section 8.1.2, Saving the Error Log Contents.

8.3.3 Error Reports by Error Type

The elp program normally produces error reports for all errors that have been recorded. You can limit an error report, however, to errors of a single specified type.

To limit the report to a single error type, use the elp program and specify the appropriate error type mnemonic. In response to a shell prompt, type a command sequence in the format:

```
$ elp -et[#]
```

et Specifies the appropriate error type mnemonic.

Specifies the unit number for block devices only (optional).

The following is a list of the error type mnemonics:

Mnemonic	Device/Type
hk	RK06/7
hj	RM02/3/5, RP04/5/6, ML11 (3rd RH Controller)
hm	RM02/3/5, RP04/5/6, ML11 (2nd RH Controller)
hp	RM02/3/5, RP04/5/6, ML11 (1st RH Controller)
hs	RS03/4
ht	TM02/3
hx	RX02
mp	Memory Parity
ra	RA60/RA80/RA81
rc	RC25
rd	RD51/RD52
rk	RK05
rl	RL01/2
rp	RP02/3
rx	RX50
sd	Shutdown
si	Stray Interrupt
su	Startup
sv	Stray Vector
tc	Time Change
tm	TM11
ts	TS11/TSV05/TU80/TK25

For example, the following command sequence limits the error report to those for an RK06/7, unit 0:

```
$ elp -hk0
```

8.3.4 Error Reports for Hard and Soft Errors

The elp program normally produces error reports for all errors that have been recorded. You also can limit the contents of the error report to either soft (recoverable) errors or hard (unrecoverable) errors.

To limit the report to soft errors only, use the elp program and specify the -r option. In response to a shell prompt, type this command sequence:

```
$ elp -r
```

To limit the report to hard errors only, use the elp program and specify the -u option. In response to a shell prompt, type this command sequence:

```
$ elp -u
```

8.3.5 Error Reports by Date and Time

The elp program normally produces error reports for all errors that have been recorded. You also can limit the report to those errors that occurred during a specified date and time range.

To limit the report to those errors that occurred during a designated period, use the elp program and specify the -d option and appropriate time range. In response to a shell prompt, type a command sequence in the format:

```
$ elp -d sd ed
```

sd Specifies the starting date and time.

ed Specifies the ending date and time.

Both the starting and ending times have the format:

yymmddhhmmss

yy Indicates the year (00-99).

mm Indicates the month (01-12).

dd Indicates the day (01-31)

hh Indicates the hour (00-23).

mm Indicates the minute (00-59).

ss Indicates the seconds (00-59).

Because there are no default values, you must type all digits when specifying either a starting or ending time. For example, the following command sequence limits the report to those errors that occurred between 12:00 am, January 1, 1984 and 11:59 pm, January 2, 1984.

```
$ elp -d 840101000000 840102115959
```

8.3.6 Multiple Options

You can specify several elp options on a single command line. When multiple options are specified, the elp program produces a report that contains only those errors which match the defined set of conditions. For example, the following command sequence combines five options:

```
$ elp -b -u -hk2 -d 840902120000 840902130000 el_sep.2
```

This command produces a report that provides only brief descriptions of each error and that used the data contained in the named file (el_sep.2) for unrecoverable errors on RK06/7, drive 2 occurring between 1200 and 1300 hours on September 2, 1984.

8.4 Sample Error Reports

The next nine sections provide descriptions of a particular error record, a corresponding sample error report, and an explanation of the reported data.

8.4.1 Error Records -- Common Header

The common header for each error log record has the format:

Sequence # ERROR TYPE Date and Time

In addition to this header, each error record may also print additional information.

8.4.2 Block I/O Device Error Record

The system logs a block I/O device error record every time that a disk or tape encounters either a hard or soft error. Block I/O device errors usually are the most common type of errors found in the error log file.

In addition to the common header, each block I/O device error record lists:

- Major/minor device number
- Generic device name
- Unit number
- Controller's CSR address
- Number of entries attempted
- Error type (recoverable or unrecoverable)
- Names of other active devices (if any)

This information is followed by the contents of the device's hardware registers and six lines of data:

- I/O buffer's physical address
- Transfer required the UNIBUS map (yes or no)
- Transfer size
- Type of I/O operation (read or write)
- Logical block number where the transfer started
- Type of operation (buffered or physical)

The logical block number is relative to the start of the file system. On partitioned disks, the start of a file system may or may not be the start of the disk. When a buffered I/O operation is indicated, the I/O error most likely occurred with a buffer in the ULTRIX-11 buffer pool. When a physical I/O operation is indicated, the I/O error most likely occurred directly between a buffer in the user process and the device. For further information, read Chapter 1, ULTRIX-11 I/O System.

The following is a sample block I/O error record:

```
Sequence 68       BLOCK I/O DEVICE ERROR - Tue Aug  3 10:50:39 1982

     Major/Minor Device      23/15
     Device Type             (first RH11/RH70) RP04/5/6, RM02/3/5, ML11
     Unit Number             1
     Device CSR Address      176700

     Retry Count             1
     Error Diagnosis         RECOVERED via RETRY
     Block Device Activity:
                             (first RH11/RH70) RP04/5/6, RM02/3/5, ML11

     Device Register Contents at time of First Error

            RMCS1            144670  SC TRE DVA A16 RDY <READ DATA>
            RMWC             177000  word count -512
            RMBA             136154
            RMDA             001401  track 3 sector 1
            RMCS2            000101  IR U0
            RMDS             150700  ATA ERR MOL DPR DRY VV
            RMER1            010000  DTE
            RMAS             000002  ATA1
            RMLA             000000
            RMDB             000000
            RMMR1            000010
            RMDT             024024  MOH dual ported RM03
            RMSN             114002  serial number 9802
            RMOF             010000  FMT16
            RMCA             000174  cylinder 124
            RMHR             000174  cylinder 124
            RMMR2            011777
            RMER2            000000
            RMPOS            004066
            RMPAT            000000
            RMBAE            000005  A18 A16
            RMCS3            000000

     Physical Buffer Start Address  1236154
     UNIBUS Map used for transfer?  NO
     Transfer size in bytes         1024
     Transfer Type                  READ
     Block in logical file system   0
     I/O Operation type             PHYSICAL
```

8.4.3 Memory Parity Record

The system logs a memory parity record every time that a memory parity error occurs while executing a user process. When the ULTRIX-11 system is in kernel mode, a memory parity error usually causes the system to crash. For further information, read Section 10.2.15, panic: parity

In addition to the common header, each memory parity error record lists the contents of the available memory hardware registers.

The following is a sample memory parity record:

```
Sequence 8        MEMORY PARITY ERROR - Sun Mar 28 11:10:10 1982


                  Memory System Register contents

                  MLEA      33034
                  MHEA      40036
                  MSER      104014
                  MSCR      3
```

8.4.4 RX50/RD51/RD52/RA60/RA80/RA81/RC25 Disk Error Record

The ULTRIX-11 operating system communicates with RX50/RD51/RD52/RA60/RA80/RA81/RC25 disks using the Mass Storage Control Protocol (MSCP). The MSCP disks can produce multiple error log entries for a single error. The end message always reports the final status of the error and indicates an MSCP command has completed. Asynchronous messages (datagrams) report status information about error retry attempts. These datagrams can arrive before and/or after the command end message.

The only method for associating these datagrams with the proper MSCP command is to scan the error log report manually for an MSCP end message that has the same command reference number and error log reference number as the datagram. The command end message and any associated datagrams should be grouped reasonably close together in the error log report.

8.4.5 Shutdown Record

The system logs a shutdown record each time that error logging is manually disabled and every time that multiuser mode is shut down.

Each shutdown record simply lists the common header information: the date and time of the shutdown.

The following is a sample shutdown record:

```
Sequence 2        ERROR LOGGING SHUTDOWN - Thu Feb  6 10:04:24 1982
```

8.4.6 Startup Record

The system logs a start-up record each time it enables error logging. Normally, this occurs during multiuser startup (transition between single-user mode and multiuser mode). Each start-up record, therefore, provides information about

a system startup.

In addition to the common header, each start-up record provides a brief system profile:

- Version of ULTRIX-11 software
- Type of processor used
- Processor features that are enabled
- Devices configured into that ULTRIX-11 kernel

The following is a sample start-up record:

```
Sequence 0      ERROR LOGGING STARTUP - Fri Jul 16 15:30:26 1982

    System Profile:

            ULTRIX-11 (V2.0) Operating System
            11/70 Processor
            UNIBUS Map Enabled
            22 Bit Mapping Enabled
            Kernel D Space Enabled
            User D Space Enabled

    Configured with:

            (UDA50) - RA60/RA80/RA81
            (RUX1) - RX50
            (RL11) - RL01/2
            (RH11/RH70 - TM02/3) - TU16/TE16
            (first  - RH11/RH70) - RP04/5/6, RM02/3/5, ML11
            (second - RH11/RH70) - RP04/5/6, RM02/3/5, ML11
             CONSOLE
             LP11
             DH11
             DZ11/DZV11/DZQ11
```

8.4.7 Stray Interrupt Record

The system logs a stray interrupt record every time that an interrupt occurs from an inactive configured device (the interrupt was not expected).

In addition to the common header, each stray vector record lists the device's UNIBUS CSR address and the names of the other devices that were active at that time. The following is a sample stray interrupt record:

```
Sequence 67     STRAY INTERRUPT - Sun Dec  7 14:56:08 1982

                From Controller at        177440
                Block Device Activity:    NONE
```

NOTE

If the reported address is less than 1000 (octal), the stray interrupt really is a stray vector from the unused vector address in a user-written device driver.

8.4.8 Stray Vector Record

The system logs a stray vector record every time that a vector through an unused vector location occurs. This indicates either an interrupt from an unconfigured device or a device that interrupted through the wrong vector address.

In addition to the common header, each stray vector record lists the stray vector address and the names of the other active devices at that time. This is a sample stray vector record:

```
Sequence 45     STRAY VECTOR - Mon Dec  8 16:40:08 1982

                From Vector Address        0340
                Block Device Activity:
                                           (RP11) - RP03
```

8.4.9 Time Change Record

The system logs a time change record every time that you use the date command to change the system's date and time. The purpose of the time change record is to prevent possible confusion caused by this change.

Instead of the normal date and time in the common header, each time change record lists two times: changed from and changed to. The following is a sample time change record:

```
Sequence 353    TIME CHANGE ***** FROM Sat Sep  4 15:54:24 1984
                                  TO  Sat Sep  4 15:54:00 1984
```

Chapter 9

ULTRIX-11 Crash Dump Analysis Facility

When an unscheduled event interrupts normal operations, the ULTRIX-11 system voluntarily crashes. Normally, the first indication of a system crash is the loss of terminal response. Another indication of a system crash is the system panic message that was written to the console. Before it crashes, the system prints a panic message at the console. For further information, read Chapter 10, ULTRIX-11 Error Messages.

The remaining sections of this chapter discuss the procedure that you should use to determine the cause of your system crash. Specifically, these sections discuss:

- Checking the console switch display register
- Gathering system status information
- Making a crash dump
- Copying the dump to a core file
- Using the cda program to analyze the core file
- Using the adb program to print kernel structures

9.1 Checking the Console Switch Display Register

The PDP-11/45, PDP-11/55, and PDP-11/70 processors are equipped with a console display register. The ULTRIX-11 operating system uses this register to display system status information. While the system is running, you can use this register to display the contents of a specified address (real time).

Because it remains active even when a system crash occurs, the console display function is very useful for initial crash analysis. You can use it to display this information dynamically without halting the processor:

- Contents of memory
- Processor error registers
- Processor status registers
- Device error registers
- Device status registers

To examine an address, first load the desired address into the console switches. Then, set the console DATA display select switch to the "display register" position.

The system displays the contents of the specified address in the display register. It uses bit zero of the specified address to determine how the address is mapped. The system maps odd addresses to user data space and even addresses to kernel data space. If you select a nonexistent address, the system displays a series of ones (1s).

As long as the system clock is enabled, the console display function remains active. On each clock interrupt (50/60 per second), the ULTRIX-11 system examines the address contained in the console switches and writes its contents to the display register. If you select a nonexistent address, the scanning rate slows to once every two seconds. This change prevents the performance degradation that would result if the bus time-out traps were allowed to occur on each clock interrupt.

NOTE

If the contents of the hardware device registers are subsequently altered, you should not use the console display function to examine them. This most often occurs with disk and communications device silo data buffer registers. Accessing these registers with the display function diverts silo data from its intended destination to the display register.

9.2 Gathering System Status Information

When your ULTRIX-11 system crashes, you should examine and record the status of your hardware:

- Console display lights
- Processor error registers
- Memory error registers
- Device error registers
- Device status registers

After examining your hardware, you should read any messages that were written to the system console. For further information about interpreting system messages, read Chapter 10, ULTRIX-11 Error Messages.

9.3 Making an ULTRIX-11 Crash Dump

To make an ULTRIX-11 crash dump, write an image of physical memory to the crash dump device. For most configurations, the crash dump device is tape. However, the crash dump device can also be the swap area of the system disk or RX50 diskettes. You determine the crash dump device during a system generation.

9.3.1 Writing a Crash Dump to Tape

To write a crash dump to tape, you first should ready the tape unit. Specifically, mount a write-enabled tape on unit 0. Then, make sure that the tape is at load point (BOT) and that the unit is on-line.

Once the tape unit is ready, you should:

- Halt the processor
- Load address 1000 (octal)
- Start the processor

When loaded and running, the crash dump code automatically writes out physical memory to tape. The crash dump code proceeds until it either has written all memory out to the tape or has encountered an error.

To indicate the completion of a successful dump, the processor halts once the tape stops. To indicate an unsuccessful dump, however, the processor hangs in a tight loop once the tape stops. Any error can stop the crash dump code. To determine what error occurred, examine the tape status and error registers.

To restart the dump, you should:

- Manually rewind the tape
- Reload address 1000 (octal)
- Restart the processor

NOTE

Even if the dump does not complete successfully, you should attempt a crash dump analysis with the partially written dump tape. Enough memory may have been written out to allow for at least a partial crash analysis.

9.3.2 Writing a Crash Dump to the System Disk

To write a crash dump to the system disk, you first should ready the disk unit. Once the system disk is ready, you should:

- Halt the processor
- Load address 1000 (octal)
- Start the processor

When loaded and running, the crash dump code writes out memory to the swap area on the system disk (starts at block 300). The crash dump proceeds until either all of memory is dumped or the swap area is full.

To indicate the completion of a successful dump, the processor halts. To indicate an unsuccessful dump, the processor hangs in a tight loop when an error occurs.

To restart the dump, you should:

- Halt the processor
- Reload address 1000 (octal)
- Restart the processor

NOTE

Even if the dump does not complete successfully, you should attempt a crash dump analysis with the partially written dump. Enough memory may have been written out to allow for at least a partial crash analysis.

9.3.3 Writing a Crash Dump to RX50 diskettes

To write a crash dump to RX50 diskettes, you first should load a write-enabled diskette in RX50 unit 2. Once the diskette is ready, you should:

- Halt the processor
- Load address 1000 (octal)
- Start the processor

When loaded and running, the crash dump code writes out memory to the diskette. The crash dump proceeds until either all of memory is dumped or another diskette is needed to continue the dump.

When a diskette is full, the crash dump code will print an asterisk on the console and halt. You then should remove the

current dump diskette and load another diskette into RX50 unit 2. Once the diskette is loaded, restart the processor. Continue until the crash dump code halts without printing an asterisk on the console.

To indicate the completion of a successful dump, the processor halts. To indicate an unsuccessful dump, the processor hangs in a tight loop when an error occurs.

To restart the dump, you should:

- Halt the processor
- Reload address 1000 (octal)
- Restart the processor

NOTE

Even if the dump does not complete successfully, you should attempt a crash dump analysis with the partially written dump. Enough memory may have been written out to allow for at least a partial crash analysis.

9.4 Copying the Crash Dump to a Core File

Before you can analyze the crash dump, you must copy it into a file (normally, /usr/crash/core).

To copy the crash dump to /usr/crash/core, first reboot the system to single-user mode. For more specific information, read Steps 1-3 of Section 3.3, Autobooting from the System Disk.

NOTE

To assure that the crash dump is not inadvertently destroyed, you should not invoke multiuser mode until after you have copied the crash dump to a core file.

Next, determine if there is enough space in /usr for the crash dump. To determine the core file size, calculate two disk blocks for every 1K bytes of memory that were dumped to the crash dump device. For example, if 256K bytes of memory were dumped to the crash dump device, the core file size would be 512 disk blocks. Then, determine if there is sufficient space on the /usr file system for a file of this size.

To determine if there is sufficient space on the /usr file system, print the /etc/fstab file to locate the logical disk name on which the /usr file system is mounted. Having determined the logical disk name, use the df command to display the amount of disk free space that is available.

The following sequence of commands is for displaying the amount of free space on a RL02-based /usr file system:

```
# cat /etc/fstab
/dev/rl00:/:rw
/dev/rl01:/usr:rw

# df /dev/rl01
/dev/rl01 3550
```

The next sequence of commands is for determining the amount of free space on a RD51-based /usr file system:

```
# cat /etc/fstab
/dev/rd00:/:rw
/dev/rd01:/usr:rw

# df /dev/rd01
/dev/rl01 6725
```

If the /usr file system has enough free space available, then copy the dump to the /usr/crash/core file. If the /usr file system does not have sufficient space available, copy as much of the crash dump to the core file as the /usr file system allows.

9.4.1 Using the ccd Program

To copy the crash dump from the crash dump device to /usr/crash/core, you first should mount the /usr file system and change to the crash directory. In response to the superuser prompt, type this sequence of commands:

```
# cd /
# /etc/mount -a
# cd /usr/crash
```

To copy the core file from the crash dump device to /usr/crash/core, use the ccd program. In response to the superuser prompt, type:

```
# ccd
```

The ccd program is interactive and automatically prompts you for all the required information. If you need help with any prompt, type ? and press the <RETURN> key. To use the default response for any prompt, press the <RETURN> key.

To access the name list and determine current system information, the ccd program prompts for the name of the file that contains the current running kernel. The ccd program uses the name list in this file to determine the crash dump device. Then, it prompts for the name of the file that contains the kernel that was running at the time of the crash. By default, these files both are normally named /unix.

The ccd program also prompts for information about the crash dump device. If no responses are given, it uses the information from the name list as the default answers. The ccd program then prompts for the name of the core file. If no response is given, it uses /usr/crash/core (default).

The ccd program also prompts for the number of blocks to copy to the core dump file. If no response is given, it copies the entire dump (default). To copy other amounts of the core file, first calculate two blocks for every 1K bytes of dumped memory and then type the proper number to the ccd prompt. The first 256K bytes of memory may be enough to allow you to analyze the crash dump.

After the ccd program has copied the crash dump to the core file, unmount all file systems. In response to the superuser prompt, type this sequence of commands:

```
# cd /
# /etc/umount -a
```

For further information, read mount(1M) in the ULTRIX-11 Programmer's Manual, Volume 1.

9.4.2 Saving Core Files

Because a system core file can consume a large amount of disk space, the /usr file system usually has space for only one core file. You can keep more than one core file by copying it back either to tape or to another file system and by removing it from the /usr/crash directory.

Once you have analyzed the core file, you may want to keep an archive copy of it. When you keep a copy of a core file, you also must save a copy of the ULTRIX-11 kernel that was running at the time of the crash. The kernel file (normally named /unix) contains the system name list that is needed by the cda program to analyze the crash dump.

To copy core files back to tape, use this procedure:

- Mount the /usr file system
- Change to the crash directory
- Make a copy of the kernel file
- Rename the core file
- Copy these files with the tar command
- Remove both the core and kernel files

For example, the following sequence of commands prepares and copies both the core file and a copy of the kernel to tape:

```
# cd /usr/crash
# cp /unix unix.may1
# mv core core.may1
# tar cvb 20 ./unix.may1 ./core.may1
# rm unix.may1 core.may1
```

To copy core files back to another file system, use this procedure:

- Mount the target file system
- Make a copy of the core and kernel files
- Remove both the core and kernel files
- Unmount the target file system

For example, the following sequence of commands prepares and copies both the core file and a copy of the kernel to the /mnt file system:

```
# /etc/mount /dev/rl17 /mnt
# cp /unix /mnt/unix.may1
# cp /usr/crash/core /mnt/core.may1
# rm /usr/crash/core
# /etc/umount /dev/rl17
```

NOTE

DIGITAL recommends that, when selecting names for the core and kernel files, you should use either the date of the crash, as in the example above, or some other mnemonic to identify them.

9.5 Using the cda Program

The cda program may be used either in single-user or in multiuser mode to extract, format, and display information from a core file.

When in single-user mode, mount the /usr file system and change to the crash directory. Then, use the cda program to analyze the core file. In response to the superuser prompt, type this sequence of commands:

```
# /etc/mount -a
# cd crash
# cda [-options namelist corefile]
```

When in multiuser mode, log in to the root (superuser) account. Then, change to the /usr/crash directory and use the cda command to analyze the core file. In response to the superuser prompt, type the following sequence of commands:

```
# cd crash
# cda [-options namelist corefile]
```

To analyze a crash dump, specify the desired options and, as input, the ULTRIX-11 name list and the core file.

The cda program provides on-line help information. To obtain help, use the cda program and specify the -h option. In response to the superuser prompt, type this command sequence:

```
# cda -h
```

The next 10 sections discuss the common cda header message and the 9 cda options. Although the cda command can be run without specifying the kernel name list or the core file, the examples provided in these sections do specify these optional command arguments.

9.5.1 Common Header Message

As a common header for all output, the cda program prints:

- ULTRIX-11 software version number
- Whether kernel is split I and D
- Processor type
- Last 128 characters printed at the console

The ULTRIX-11 kernel saves the last 128 characters printed at the console in an internal message buffer. In the event of a system crash, this buffer contains the panic and system error messages that pertain to the crash.

9.5.2 -b Option

To display the state of the ULTRIX-11 I/O buffer pool at the time of the crash, use the cda command and specify the -b option.

The generated display indicates the following information:

- Physical memory address
- Number of I/O operations
- Byte count of the last I/O operation
- Logical block number of the last I/O operation
- Header error status byte
- Name of the device owning the buffer
- Header status byte flags

For example, this command sequence produces the following buffer pool status report:

```
        # cda -b /unix /usr/crash/core


Crash Dump Analysis of ULTRIX-11 V2.0 Kernel
Split I & D overlay version on a PDP11/70

****** Last 128 characters of console messages ******

ULTRIX-11 Kernel V2.0

realmem = 1048576
buffers = 14848
usermem = 937792

SV 324

ka6 = 6001
aps = 140700
pc = 41604 ps = 30340
trap type 1
panic: trap

****** I/O buffer pool usage ******

Buffer Pool Status

Buffer    I/O ops  Byte    Logic
Address   per buf  count   blkno   err   device        flags
-------------------------------------------------------------
015232    4147     01000   0630    0     hp00          RD|DN
016232    2489     00      00      0     system        WRT|BSY
017232    1385     01000   020     0     hp00          WRT|DN
020232    562      01000   023     0     hp00          WRT|DN
021232    750      01000   021200  0     system        WRT|BSY
022232    379      01000   0310    0     system        WRT|BSY
```

```
023232    250     01000    043      0    hp00          WRT|DN
024232    70      01000    0606     0    hp00          RD|DN
025232    95      01000    052      0    system        WRT|BSY
026232    35      01000    017      0    hp00          WRT|DN
027232    239     01000    073      0    hp00          WRT|DN
030232    250     01000    011132   0    hp00          RD|DN
031232    419     01000    064      0    hp00          WRT|DN
032232    286     01000    060      0    hp00          WRT|DN
033232    222     01000    013532   0    hp00          RD|DN
034232    212     01000    072      0    hp00          WRT|DN
035232    206     01000    014144   0    hp00          RD|DN
036232    204     01000    01       0    hp03          WRT|DN
037232    216     01000    016      0    hp03          WRT|DN
040232    202     01000    024      0    hp00          WRT|DN
041232    220     01000    063      0    hp00          WRT|DN
042232    37      01000    036      0    hp00          WRT|DN
043232    393     01000    021736   0    system        WRT|BSY
044232    216     01000    054      0    hp00          WRT|DN
045232    0       01000    067      0    hp00          WRT|DN
046232    1       01000    055      0    hp00          WRT|DN
047232    4       01000    037      0    hp00          WRT|DN
050232    1       01000    035      0    hp00          WRT|DN
051232    0       01000    062      0    hp00          WRT|DN
```

9.5.3 -e Option

To scan the kernel's internal buffers for error messages that were not logged at the time of the system crash, use the cda command and specify the -e option.

If the system crashes while logging an error, it is possible that normal system operation could stop before that error is written to the error log file. When this occurs, the error information usually is stored in the internal error message buffer. In many cases, this error information is the only indicator of the cause of the system crash.

When the -e option is specified, the cda program locates unlogged errors and passes them to the error log print program, elp. In addition, the cda program examines the internal error buffer for each block I/O device and reports on the last error logged by that device. For further information about elp error reports, read Section 8.4, Sample Error Reports.

For example, this command sequence produces the following report of unlogged errors.

```
# cda -e /unix /usr/crash/core


Crash Dump Analysis of ULTRIX-11 V2.0 Kernel
Split I & D overlay version on a PDP11/70
```

```
****** Last 128 characters of console messages ******

ULTRIX-11 Kernel V2.0

realmem = 2097152
buffers = 73728
usermem = 1929024
panic: IO err in swap

****** Analysis of unlogged errors ******

Error log buffer at     007102
Buffer size in words  = 000512
Buffer input  pointer = 007630
Buffer output pointer = 007630

****** No unlogged errors in kernel buffer ******

****** Last error on each Block I/O device ******

ULTRIX-11 SYSTEM ERROR REPORT, taken on - Tue Sep 7 15:32:42 1982

Error type limitations: [ NONE ]

Input taken from        /tmp/cda_el.18639

REPORTED DATE/TIME RANGE: [ first error thru last error ]

        Wed Dec 31 19:00:00 1969 thru Thu Apr 22 13:55:16 1982

TOTAL startup           = 0

TOTAL shutdown          = 0

TOTAL time change       = 0

TOTAL stray interrupts  = 0

TOTAL stray vectors     = 0

TOTAL memory parity     = 0

TOTAL block I/O errors  = 1

(first  - RH11/RH70) - RP04/5/6, RM02/3/5, ML11

 UNIT 0

        Hard errors             = 1
        Soft errors             = 0
        (ECC   recovered)       = 0
        (RETRY recovered)       = 0

Sequence 0  BLOCK I/O DEVICE ERROR - Thu Apr 22 13:55:16 1982
```

```
  Major/Minor Device  9/1
  Device Type         (first RH11/RH70) RP04/5/6, RM02/3/5, ML11
  Unit Number         0
  Device CSR Address  176700

  Retry Count         29
  Error Diagnosis     NOT RECOVERED
  Block Device Activity:
                      (first RH11/RH70) RP04/5/6, RM02/3/5, ML11

  Device Register Contents at time of First Error

          RPCS1   145660  SC TRE DVA A17 A16 RDY <WRT DATA>
          RPWC    161450  word count -7384
          RPBA    140220
          RPDA    001006  track 2 sector 6
          RPCS2   020300  UPE OR IR
          RPDS    010700  MOL DPR DRY VV
          RPER1   000000
          RPAS    000000
          RPLA    000640
          RPDB    000000
          RPMR    000400
          RPDT    020022  MOH RP06
          RPSN    003471  serial number 0739
          RPOF    110000  SGCH FMT22
          RPCA    000030  cylinder 24
          RPCC    000030  cylinder 24
          RPER2   000000
          RPER3   000000
          RLPOS   000000
          RLPAT   000000
          RPBAE   000037  A20 A19 A18 A17 A16
          RPCS3   062000  DBEOW DPEEW DBL

  Physical Buffer Start Address   7627600
  Unibus Map used for transfer ?  NO
  Transfer size in bytes          51904
  Transfer Type                   WRITE
  Block in logical file system    0
  I/O Operation type              PHYSICAL
```

In this report, the common header also includes the following panic message:

```
panic: IO err in swap
```

This error message indicates an unrecoverable I/O error occurred during a swap operation.

The next item in the report is an analysis of unlogged errors that identifies the following information about the error log buffer:

- Address in memory
- Size
- I/O pointer values

In the report, there were no unlogged errors in the error log buffer. The report, however, does print out an error report for the last error on each block I/O device.

By examining this sample report, you can determine that there was a UNIBUS parity error (UPE) in the control and status two (hpcs2) register of the system disk. This report further indicates that a memory parity error occurred in the process image that the system was swapping. The contents of the bus address and bus-address-extension registers point to the address of the faulty memory location. The system crashed before writing any of this information to the error log file.

9.5.4 -g# Option

To display the Ublock information either for all processes or for an individual process, use the cda command and specify the -g option.

If the address (octal) of the desired process is specified, the cda command displays the Ublock information for that process only. If no address is specified, however, the cda command displays the Ublock information for all processes.

For example, this command sequence produces the following Ublock report for the specified process.

```
# cda -g4040 /unix /usr/crash/core


Crash Dump Analysis of ULTRIX-11 V2.0 Kernel
Split I & D overlay version on a PDP11/70

****** Last 128 characters of console messages ******

ULTRIX-11 Kernel V2.0

realmem = 524288
buffers = 73728
usermem = 343232

ka6 = 16253
aps = 140712
pc = 27106 ps = 30000
trap type 0
panic: trap
```

```
****** Ublock information ******

/unix /usr/crash/core  at 404000   Thu Jul 26 10:43:35 EDT 1984

u_comm = sh
u_base = 43357
u_count = 0
u_offset = 0
u_rsav
30010  136344 105346 140746 140732 22570  0
u_inemt = 0
u_fpsaved = 1
u_fps.u_fpsr = 200
u_fps.u_fpregs
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
u_segflg = 0
u_error = 0
u_uid = 0
u_gid = 1
u_ruid = 0
u_rgid = 1
u_procp = 105346
u_ap = 141456
u_r.r_val1 = 23024
u_r.r_val2 = 150
u_cdir = 60004
u_rdir = 0
u_dbuf = sysxr
u_dirp = 0
u_dent.d_name = sysxr
u_pdir = 0
u_uisa
0      200    20     0      0      0      0      177731
u_udsa
0      200    20     0      0      0      0      177731
u_uisd
77422  72022  31006  0      0      0      0      65016
u_udsd
77422  72022  31006  0      0      0      0      65016
u_pofile
0      0      0      0      0      0      0      0      0      0
1      1      0      0      0      0      0      0      0      0
u_ofile
53470  53470  53470  0      0      0      0      0      0      0
0      53470  0      0      0      0      0      0      0      0
u_arg
0      43356  1      2      0
u_tsize = 365
u_dsize = 63
u_ssize = 26
u_qsav
6042   35270  0      141034 141004 6122   0
u_ssav
131    105346 105742 140764 140746 23602  0
```

```
u_signal
0       0       34742   1       0       0       0       0       0       0
0       35006   0       0       35022   1       0       0       1       0
0       0       0       0       0       0       0       0       0       0
0       0
u_utime = 12
u_stime = 176
u_cutime = 6
u_cstime = 54
u_ar0 = 141050
u_prof.pr_base = 0
u_prof.pr_size = 0
u_prof.pr_off = 0
u_prof.pr_scale = 0
u_intflg = 0
u_sep = 0
u_ttyp = 13550
u_ttyd = 0
u_exdata.ux_mag = 410
u_exdata.ux_tsize = 36500
u_exdata.ux_dsize = 3116
u_exdata.ux_bsize = 512
u_exdata.ux_ssize = 0
u_exdata.ux_entloc = 0
u_exdata.ux_unused = 0
u_exdata.ux_relflg = 1
u_start = 3317344430
u_acflag = 2
u_fpflag = 0
u_cmask = 2
u_fperr.f_fec = 0
u_fperr.f_fea = 0
u_ovdata.uo_curov = 0
u_ovdata.uo_ovbase = 0
u_ovdata.uo_dbase = 0
u_ovdata.uo_ov_offst
0       0       0       0       0       0       0       0
u_ovdata.uo_nseg = 0
u_eosys = 0
```

9.5.5 -m Option

To display a map of memory usage at the time of the crash, use the cda command and specify the -m option.

The generated memory map indicates the areas of memory that were occupied by each process in the system and that are not in use. In addition to the common header, the generated report displays the contents of memory starting at address 0 and ending at the end of physical memory.

For example, this command sequence produces the following memory usage report:

```
# cda -m /unix /usr/crash/core

Crash Dump Analysis of ULTRIX-11 V2.0 Kernel
Non-split I & D overlay version on a PDP11/40

****** Last 128 characters of console messages ******

ULTRIX-11 Kernel V2.0

realmem = 196608
buffers = 12800
usermem = 103296


****** Map of memory usage ******

Total Memory = 179 Kbytes       Tue Aug 28 10:26:20 EDT 1984
  Addr    Command         Pid  Size   Ublock    Text      Data     Stack
-------------------------------------------------------------------------
    0      ULTRIX-11           79488
    v
0233200   swapper          0   1024   0233200
    v
0235200   *BUFFERS*            12800
    v
0266200   *FREE MEMORY*        2112
    v
0272300   init             1   3968   0272300 *0302100* 0274300 0277400
    v                                   1024      3264    1600    1344
0302100   *TEXT SEGMENT*       3264
    |      init
    v
0310400   sh                 3   5632   0310400 *0350500* 0312400 0320700
    v                                   1024     15680     3264    1344
0323400   *FREE MEMORY*        2304
    v
0330000   elc                2   8512   0330000  0332000  ------> 0346000
    v                                   1024     6144              1344
0350500   *TEXT SEGMENT*       15680
    |      sh
    v
0407200   sh                 119 6208   0407200 *0350500* 0411200 0420500
    v                                   1024     15680     3776    1408
0423300   *FREE MEMORY*        6208
    v
0437400   memstat            120 22144  0437400  0441400  ------> 0510000
    v                                   1024     19712             1408
0512600   *FREE MEMORY*        27264
    v
0547000
```

9.5.6 -px Option

To display the status of each process that was active as well as the contents of the system's internal tables that were in use at the time of the crash, use the cda command and specify the -p option.

The x indicates the additional option modifiers that the cda program is to use in generating the desired output. The following option modifiers (and their corresponding ps and pstat command sequence) may be used:

Option	Command	Meaning
s	ps -alx	(process status)
p	pstat -p	(process table active slots)
pa	pstat -pa	(process table all slots)
i	pstat -i	(inode table)
f	pstat -f	(open files)
x	pstat -x	(text table)
t	pstat -t	(terminal status)
u#	pstat -u addr	(dump U block of process at addr)
paifxtu#		(all of the above)

For further information, read ps(1) and pstat(1M) in the ULTRIX-11 Programmer's Manual, Volume 1.

For example, this command sequence produces the following process and table status report:

```
        # cda -ps /unix /usr/crash/core


Crash Dump Analysis of ULTRIX-11 V2.0 Kernel
Split I & D overlay version on a PDP11/70

****** Last 128 characters of console messages ******


ULTRIX-11 Kernel V2.0

realmem = 1048576
buffers = 73728
usermem = 855808

SV 324

SV 324

SV 324

SV 324

ka6 = 6001
```

```
aps = 140700
pc = 41604 ps = 30340
trap type 1
panic: trap

****** Status of active processes ******

CAUTION , if the process is swapped (F = 0), being swapped (F = 11)
         , then TTY, TIME and CMD could be erroneous !

  F S UID PID LPID CPU PRI NICE ADDR  SZ  WCHAN TTY TIME CMD
  3 S   0   0    0   0   0   20 2713   2   6662 ?    1:26 swapper
  1 S   0   1    0   0  30   20 2774   8 112034 ?    0:01 /etc/init
101 S   0   2    1   0   5    0 3337  17   7106 co   0:00 /etc/elc
  1 R   0  49   37  20  55   24 5475  25        co  53:55 cmxr dh0
  1 S   0  37    1   0  28   20 4255  11  11350 co   0:02 -sh
  1 R   0  24    1   0  40   20 3214   5        ?    0:10 /etc/update
  1 R   1  28    1   0  40   20 4744  10        ?    0:04 /etc/cron
  1 R   0  50   37 225  67   24 6001  17        co  79:52 cmxr dz0
```

9.5.7 -q Option

To display the contents of the block I/O header queues (including forward and backward pointers) at the time of the system crash, use the cda command and specify the -q option.

Those buffers that are used for superblocks are listed. The generated report also includes all device queues. To verify that all listed forward and backward pointers for the free list and device queues are correct, a consistency check is done, and the status is reported.

For example, this command sequence produces the following block I/O header queue report:

```
# cda -q /unix /usr/crash/core
```

This sample header queue report presents only the first portions (superblock and free list) of the printout:

```
Crash Dump Analysis of ULTRIX-11 V2.0 Kernel
Non-split I & D overlay version on a PDP11/40

****** Last 128 characters of console messages ******


ULTRIX-11 Kernel V2.0

realmem = 196608
buffers = 12800
usermem = 103296


********** Buffer Headers **********
```

```
25 buffers in system

Buffer #  0: Super Block for rp unit 0 partition 0
Buffer # 24: Super Block for rp unit 0 partition 2

********************* Free List Queue **********************

22 buffer headers in Freelist queue
Freelist consistency check: OK

bfreeli: address = 106414          buf # 18: address = 71064
=========================          =========================
b_forw  00106414 bfreeli           b_forw  00071150 buf # 20
b_back  00106414 bfreeli           b_back  00067774 rptab
av_forw 00071064 buf # 18  >       av_forw 00070576 buf # 11  >
av_back 00071150 buf # 20          av_back 00106414 bfreeli
=========================          =========================


buf # 11: address = 70576          buf # 10: address = 70544
=========================          =========================
b_forw  00067742 rktab             b_forw  00070576 buf # 11
b_back  00070544 buf # 10          b_back  00070172 buf #  1
av_forw 00070544 buf # 10  >       av_forw 00070630 buf # 12  >
av_back 00071064 buf # 18          av_back 00070576 buf # 11
=========================          =========================


buf # 12: address = 70630          buf #  2: address = 70224
=========================          =========================
b_forw  00071320 buf # 24          b_forw  00070630 buf # 12
b_back  00070224 buf #  2          b_back  00070310 buf #  4
av_forw 00070224 buf #  2  >       av_forw 00070310 buf #  4  >
av_back 00070544 buf # 10          av_back 00070630 buf # 12
=========================          =========================


buf #  4: address = 70310          buf # 17: address = 71032
=========================          =========================
b_forw  00070224 buf #  2          b_forw  00070310 buf #  4
b_back  00071032 buf # 17          b_back  00071234 buf # 22
av_forw 00071032 buf # 17  >       av_forw 00071234 buf # 22  >
av_back 00070224 buf #  2          av_back 00070310 buf #  4
=========================          =========================


buf # 22: address = 71234          buf #  1: address = 70172
=========================          =========================
b_forw  00071032 buf # 17          b_forw  00070544 buf # 10
b_back  00071266 buf # 23          b_back  00070714 buf # 14
av_forw 00070172 buf #  1  >       av_forw 00071266 buf # 23  >
av_back 00071032 buf # 17          av_back 00071234 buf # 22
=========================          =========================
```

```
buf # 23: address = 71266
==========================
b_forw  00071234 buf # 22
b_back  00071202 buf # 21
av_forw 00071202 buf # 21  >
av_back 00070172 buf #  1
==========================
```

```
buf # 21: address = 71202
==========================
b_forw  00071266 buf # 23
b_back  00070426 buf #  7
av_forw 00070714 buf # 14  >
av_back 00071266 buf # 23
==========================
```

```
buf # 14: address = 70714
==========================
b_forw  00070172 buf #  1
b_back  00070460 buf #  8
av_forw 00070426 buf #  7  >
av_back 00071202 buf # 21
==========================
```

```
buf #  7: address = 70426
==========================
b_forw  00071202 buf # 21
b_back  00070374 buf #  6
av_forw 00070374 buf #  6  >
av_back 00070714 buf # 14
==========================
```

```
buf #  6: address = 70374
==========================
b_forw  00070426 buf #  7
b_back  00070342 buf #  5
av_forw 00070746 buf # 15  >
av_back 00070426 buf #  7
==========================
```

```
buf # 15: address = 70746
==========================
b_forw  00070460 buf #  8
b_back  00071000 buf # 16
av_forw 00070460 buf #  8  >
av_back 00070374 buf #  6
==========================
```

```
buf #  8: address = 70460
==========================
b_forw  00070714 buf # 14
b_back  00070746 buf # 15
av_forw 00071000 buf # 16  >
av_back 00070746 buf # 15
==========================
```

```
buf # 16: address = 71000
==========================
b_forw  00070746 buf # 15
b_back  00070662 buf # 13
av_forw 00070662 buf # 13  >
av_back 00070460 buf #  8
==========================
```

```
buf # 13: address = 70662
==========================
b_forw  00071000 buf # 16
b_back  00067742 rktab
av_forw 00070512 buf #  9  >
av_back 00071000 buf # 16
==========================
```

```
buf #  9: address = 70512
==========================
b_forw  00070342 buf #  5
b_back  00071116 buf # 19
av_forw 00070342 buf #  5  >
av_back 00070662 buf # 13
==========================
```

```
buf #  5: address = 70342
==========================
b_forw  00070374 buf #  6
b_back  00070512 buf #  9
av_forw 00071116 buf # 19  >
av_back 00070512 buf #  9
==========================
```

```
buf # 19: address = 71116
==========================
b_forw  00070512 buf #  9
b_back  00071150 buf # 20
av_forw 00071150 buf # 20  >
av_back 00070342 buf #  5
==========================
```

```
buf # 20: address = 71150
=========================
b_forw  00071116 buf # 19
b_back  00071064 buf # 18
av_forw 00106414 bfreeli  >
av_back 00071116 buf # 19
=========================


****************** End of Free List Queue ******************
```

9.5.8 -r Option

To display the contents of the resource maps (core and swap maps) at the time of the system crash, use the cda command and specify the -r option.

To verify that all reported values are valid, a consistency check is performed, and the status is reported. If either map proves inconsistent, then all locations of both maps (including zero entries) are reported. If both maps prove consistent, however, all trailing zero entries are truncated.

For example, this command sequence produces the following resource map report:

```
        # cda -r /unix /usr/crash/core


Crash Dump Analysis of ULTRIX-11 V2.0 Kernel
Split I & D overlay version on a PDP11/70

****** Last 128 characters of console messages ******

=101,10000
err on dev 29/15
bn=97184 er=101,10000
err on dev 29/15
bn=97376 er=101,10000
err on dev 29/15
bn=97504 er=101,10000


****** Resource map contents ******

Mapsize = 105

        Core Map                        Swap Map

      Size    Address                 Size    Address
        33     003155                   10     000001
       122     003367                 5984     000021
```

```
     14141     004303                    0     000000
         0     000000          102 more zero entries
101 more zero entries
```

9.5.9 -t Option

To display all hardware trap error messages that occurred at the time of the system crash, use the cda command and specify the -t option.

When a hardware-detected trap error occurs while it is in kernel mode, the system crashes and prints:

```
panic: trap
```

When a panic trap occurs, the system saves the relevant information (such as, the contents of the hardware registers) in an internal buffer.

For example, this command sequence produces the following panic: trap analysis.

```
# cda -t /unix /usr/crash/core


Crash Dump Analysis of ULTRIX-11 V2.0 Kernel
Non-split I & D overlay version on a PDP11/24

****** Last 128 characters of console messages ******

almem = 253952
buffers = 12800
usermem = 162752

ka6 = 4135
aps = 140656
pc = 45626 ps = 30000
ovno = 4
trap type 1
panic: trap


****** Dump of panic trap error stack frame ******

General Registers at error time:
        R0      000010
        R1      045624
        R2      072164
        R3      113050
        R4      072164
        R5      140674
        SP      140646  (from previous space)
        SP      140470  (after panic trap)
```

```
Memory Management Status Registers:
        MMR0    000017
        MMR2    002240
        MMR3    000000

ka6 = 4135 aps = 140656
Updated program counter = 045626
Processor status word   = 030000
Current kernel overlay  = 4
Overlay at error time   = 4

PC      Physical
offset  address  Contents
-6      170420   004567
-4      170422   140434
-2      170424   000010
0       170426   000004
+2      170430   004737
+4      170432   036144

CPU was in KERNEL mode at the time of the trap
The error type was (trap thru location 10 - Reserved instruction)
```

The saved console error messages indicate a panic trap, type one. The report contains the contents of the following resisters:

- Processor general registers
- Memory management status registers
- Processor error register (if it exists)

Then, it displays the updated program counter, the processor status word at the time of the error, and the values of ka6 and aps. For a description of these values, read Section 10.1.14, panic: trap. Finally, it displays the instructions preceding, at, and following the updated program counter. As the last item, it indicates the type of trap. In the example above, it indicates a trap through location 10.

NOTE

The instruction immediately preceding the location pointed to by the updated PC contains a value of 10. This is a reserved instruction.

9.5.10 -u Option

To display information about disk devices that use MSCP architecture, use the cda command and specify the -u option.

The following devices use MSCP architecture:

Controller	Drive
UDA50	RA60/RA80/RA81
RQDX1	RD51/RD52/RX50
KLES1	RC25
RUX1	RX50

For example, this command sequence produces the following MSCP disk report:

```
# cda -u /unix /usr/crash/core


Crash Dump Analysis of ULTRIX-11 V2.0 Kernel
Split I & D overlay version on a PDP11/70

****** Last 128 characters of console messages ******

100000,100
err on dev 26/1
bn=71740 er=100200,200
err on dev 26/1
bn=76740 er=100000,100
err on dev 26/1
bn=18300 er=100200,200


****** Mscp disk information ******

Controller number 0:    type = UDA50A

state = 5 credits = 21 tcmax = 21 lastcmd = 6 lastrsp = 1
command queue transition interrupt flag = 0
response queue transition interrupt flag = 0

********* command descriptors:

                        virtual    physical
UDA_INT  UDA_OWN        address     address
-------------------------------------------------
   1         0        00000007054 00000020770
   1         0        00000007154 00000021070
   1         0        00000007254 00000021170
   1         0        00000007354 00000021270
   1         0        00000007454 00000021370
   1         0        00000007554 00000021470
   1         0        00000007654 00000021570
   1         0        00000007754 00000021670

********* response descriptors:

                        virtual    physical
UDA_INT  UDA_OWN        address     address
-------------------------------------------------
```

```
    1         1         00000006054 00000017770
    1         1         00000006154 00000020070
    1         1         00000006254 00000020170
    1         1         00000006354 00000020270
    1         1         00000006454 00000020370
    1         1         00000006554 00000020470
    1         1         00000006654 00000020570
    1         1         00000006754 00000020670

********** Command packets:

Packet #1:
000074 000000 101614 000000 000000 000000 000011 000000
000000 000000 000000 000000 000000 000000 000000 000000
000000 000000 000000 000000 017724 000000 000000 000000
000000 000000 000000 000000 000000 000000 000000 000000

Packet #2:
000074 000000 101614 000000 000000 000000 000011 000000
000000 000000 000000 000000 000000 000000 000000 000000
000000 000000 000000 000000 017730 000000 000000 000000
000000 000000 000000 000000 000000 000000 000000 000000

Packet #3:
000074 000000 101614 000000 000000 000000 000011 000000
000000 000000 000000 000000 000000 000000 000000 000000
000000 000000 000000 000000 017734 000000 000000 000000
000000 000000 000000 000000 000000 000000 000000 000000

Packet #4:
000074 000000 101614 000000 000000 000000 000011 000000
000000 000000 000000 000000 000000 000000 000000 000000
000000 000000 000000 000000 017740 000000 000000 000000
000000 000000 000000 000000 000000 000000 000000 000000

Packet #5:
000074 000000 101614 000000 000000 000000 000011 000000
000000 000000 000000 000000 000000 000000 000000 000000
000000 000000 000000 000000 017744 000000 000000 000000
000000 000000 000000 000000 000000 000000 000000 000000

Packet #6:
000074 000000 101614 000000 000000 000000 000011 000000
000000 000000 000000 000000 000000 000000 000000 000000
000000 000000 000000 000000 017750 000000 000000 000000
000000 000000 000000 000000 000000 000000 000000 000000

Packet #7:
000074 000000 101614 000000 000000 000000 000011 000000
000000 000000 000000 000000 000000 000000 000000 000000
000000 000000 000000 000000 017754 000000 000000 000000
000000 000000 000000 000000 000000 000000 000000 000000

Packet #8:
```

```
000074 000000 101614 000000 000000 000000 000011 000000
000000 000000 000000 000000 000000 000000 000000 000000
000000 000000 000000 000000 017760 000000 000000 000000
000000 000000 000000 000000 000000 000000 000000 000000

********** Response packets:

Packet #1:
000074 000001 101614 000000 000000 000000 000211 000003
000000 000000 000000 000000 000000 000000 000000 000000
000000 000000 000000 000000 017750 000000 000000 000000
000000 000000 017000 000400 000037 000001 000000 000000

Packet #2:
000074 000001 101614 000000 000000 000000 000211 000003
000000 000000 000000 000000 000000 000000 000000 000000
000000 000000 000000 000000 017754 000000 000000 000000
000000 000000 017000 000400 000037 000001 000000 000000

Packet #3:
000074 000001 101614 000000 000000 000000 000211 000003
000000 000000 000000 000000 000000 000000 000000 000000
000000 000000 000000 000000 017760 000000 000000 000000
000000 000000 017000 000400 000037 000001 000000 000000

Packet #4:
000074 000001 101614 000000 000000 000000 000211 000003
000000 000000 000000 000000 000000 000000 000000 000000
000000 000000 000000 000000 017724 000000 000000 000000
000000 000000 017000 000400 000037 000001 000000 000000

Packet #5:
000074 000001 101614 000000 000000 000000 000211 000003
000000 000000 000000 000000 000000 000000 000000 000000
000000 000000 000000 000000 017730 000000 000000 000000
000000 000000 017000 000400 000037 000001 000000 000000

Packet #6:
000074 000001 101614 000000 000000 000000 000211 000003
000000 000000 000000 000000 000000 000000 000000 000000
000000 000000 000000 000000 017734 000000 000000 000000
000000 000000 017000 000400 000037 000001 000000 000000

Packet #7:
000074 000001 101614 000000 000000 000000 000211 000003
000000 000000 000000 000000 000000 000000 000000 000000
000000 000000 000000 000000 017740 000000 000000 000000
000000 000000 017000 000400 000037 000001 000000 000000

Packet #8:
000074 000001 101614 000000 000000 000000 000211 000003
000000 000000 000000 000000 000000 000000 000000 000000
000000 000000 000000 000000 017744 000000 000000 000000
000000 000000 017000 000400 000037 000001 000000 000000
```

```
Controller number 1:    type = RUX1

state = 5 credits = 13 tcmax = 13 lastcmd = 7 lastrsp = 0
command queue transition interrupt flag = 0
response queue transition interrupt flag = 0

********* command descriptors:

                          virtual    physical
UDA_INT  UDA_OWN          address     address
-------------------------------------------------------
   1         0         00000011164 00000023100
   1         0         00000011264 00000023200
   1         0         00000011364 00000023300
   1         0         00000011464 00000023400
   1         0         00000011564 00000023500
   1         0         00000011664 00000023600
   1         0         00000011764 00000023700
   1         0         00000012064 00000024000

********* response descriptors:

                          virtual    physical
UDA_INT  UDA_OWN          address     address
-------------------------------------------------------
   1         1         00000010164 00000022100
   1         1         00000010264 00000022200
   1         1         00000010364 00000022300
   1         1         00000010464 00000022400
   1         1         00000010564 00000022500
   1         1         00000010664 00000022600
   1         1         00000010764 00000022700
   1         1         00000011064 00000023000

********** Command packets:

Packet #1:
000074 000000 034474 002471 000000 000000 000041 000000
001000 000000 062000 000000 000000 000000 000000 000000
000713 000000 000000 000000 022034 000000 000000 000000
000000 000000 000000 000000 000000 000000 000000 000000

Packet #2:
000074 000000 034122 002472 000000 000000 000041 000000
001000 000000 051000 000000 000000 000000 000000 000000
000714 000000 000000 000000 022040 000000 000000 000000
000000 000000 000000 000000 000000 000000 000000 000000

Packet #3:
000074 000000 101630 000000 000000 000000 000011 000000
000000 000000 000000 000000 000000 000000 000000 000000
000000 000000 000000 000000 022044 000000 000000 000000
000000 000000 000000 000000 000000 000000 000000 000000
```

```
Packet #4:
000074 000000 102510 002473 000000 000000 000042 000000
012000 000000 040000 000001 000000 000000 000000 000000
000000 000000 000000 000000 022050 000000 000000 000000
000000 000000 000000 000000 000000 000000 000000 000000

Packet #5:
000074 000000 102510 002474 000000 000000 000042 000000
012000 000000 040000 000001 000000 000000 000000 000000
000012 000000 000000 000000 022054 000000 000000 000000
000000 000000 000000 000000 000000 000000 000000 000000

Packet #6:
000074 000000 102510 002475 000000 000000 000042 000000
012000 000000 040000 000001 000000 000000 000000 000000
000024 000000 000000 000000 022060 000000 000000 000000
000000 000000 000000 000000 000000 000000 000000 000000

Packet #7:
000074 000000 102510 002476 000000 000000 000042 000000
012000 000000 040000 000001 000000 000000 000000 000000
000036 000000 000000 000000 022064 000000 000000 000000
000000 000000 000000 000000 000000 000000 000000 000000

Packet #8:
000074 000000 034122 002470 000000 000000 000041 000000
001000 000000 051000 000000 000000 000000 000000 000000
000712 000000 000000 000000 022070 000000 000000 000000
000000 000000 000000 000000 000000 000000 000000 000000

********** Response packets:

Packet #1:
000074 000000 000000 000000 000000 000000 000100 000000
000400 100200 000000 000000 000000 000000 000001 001007
100062 022545 000000 000000 001440 000000 000000 000000
000000 000000 000000 000000 000000 000000 000000 000000

Packet #2:
000074 000001 101630 000000 000000 000000 000211 000000
000400 100200 000000 000000 000000 000000 000001 001007
100062 022545 000000 000000 001440 000000 000000 000000
000056 000012 177420 000200 000000 000000 000000 000000

Packet #3:
000074 000001 102510 002473 000000 000000 000242 000000
012000 000000 040000 000001 000000 000000 000000 000000
000000 000000 000000 000000 001440 000000 000000 000000
000001 000011 177420 000200 000000 000000 000000 000000

Packet #4:
000074 000001 102510 002474 000000 000000 000242 000000
012000 000000 040000 000001 000000 000000 000000 000000
000012 000000 000000 000000 001440 000000 000000 000000
```

```
000057 000011 177406 000200 000000 000000 000000 000000

Packet #5:
000074 000001 102510 002475 000000 000000 000242 000000
012000 000000 040000 000001 000000 000000 000000 000000
000024 000000 000000 010000 000000 000000 000004 000000
000001 000011 177420 000200 000000 000000 000000 000000

Packet #6:
000074 000001 102510 002476 000000 000000 000242 000000
012000 000000 040000 000001 000000 000000 000000 000000
000036 000000 000000 000000 001440 000000 000000 000000
000000 000000 000000 000000 000000 000000 000000 000000

Packet #7:
000074 000000 000000 000000 000000 000000 000100 000000
000400 100200 000000 000000 000000 000000 000001 001007
100062 022545 000000 000000 001440 000000 000000 000000
000001 000011 177420 000200 000000 000000 000000 000000

Packet #8:
000074 000000 000000 000000 000000 000000 000100 000000
000400 120200 000000 000000 000000 000000 000001 001007
100062 022545 000000 000000 001440 000000 000000 000000
000000 000000 000000 000000 000000 000000 000000 000000

********* drive information

Controller 0, Drive 0
drive type = RA60   status = offline   unit size = 0

Controller 0, Drive 1
drive type = RA81   status = online    unit size = 891072

Controller 1, Drive 0
drive type = RX50   status = offline   unit size = 0

Controller 1, Drive 1
drive type = RX50   status = online    unit size = 800
```

9.6 Using the adb Program

You also may use the adb program and one of the scripts distributed in the /usr/crash directory to print the kernel data structures from the crash dump. To use the adb program, type:

```
$ adb /unix /usr/crash/core
```

To use one of the scripts once adb is running, type:

```
address$<script
```

The distributed script names are:

Script	Kernel Structure
buff	buffer structures
dent	directory entry
dinode	indirect block structure
filsysf	superblock structure
inodef	in-memory inode structure
procf	process structure
regf	stack print registers (relative to *u_ar0)
s	stack backtrace for kernel core dumps
sregf	same as regf, but assigns the values to the adb internal variables
textf	text structure
ttyf	tty structure
userf	user structure

NOTE

DIGITAL recommends that you use the adb program and the distributed scripts to print out the kernel data structures only if you have a detailed knowledge of kernel internals.

Chapter 10

ULTRIX-11 Error Messages

The ULTRIX-11 system prints two types of error messages: kernel-mode messages and user-mode messages. The ULTRIX-11 system prints kernel-mode messages at the console when errors occur while the kernel is running. The ULTRIX-11 system prints user-mode messages at the user's terminal when errors occur while a user process is running.

The kernel-mode messages can be further classified: system panics and system warnings. When it detects an internal error that it cannot recover from, the kernel prints a panic message at the console and voluntarily crashes. When it detects an internal error that is not serious enough to stop the system from functioning, the kernel prints a warning message at the console. In addition, the kernel prints various miscellaneous messages to indicate other error conditions.

During daily operations, you can disallow printing error messages at the console. To disallow messages at the console, press <CTRL/K>. To allow messages at the console, press <CTRL/A>.

The remaining sections of this chapter discuss:

- System message log
- System panic messages
- System warning messages
- Miscellaneous messages
- User-mode messages

10.1 System Message Log

When an error occurs while the ULTRIX-11 kernel is running, the kernel's own internal printf() routine normally prints a panic, warning, or miscellaneous message at the system console. Regardless of whether it is allowed to print to the console, the printf() function writes the same message in the system's in-memory message buffer. To save this information, the system executes the dmesg program every ten minutes. The dmesg program collects the information from the message buffer and appends it to the end of the system message log, /usr/adm/messages.

The /usr/adm/messages file contains a log of all system messages printed at the system console. DIGITAL recommends that you should review the message log regularly.

NOTE

> The size of the in-memory message buffer is limited to 128 bytes. If it writes more than 128 characters (bytes) to this buffer, the system simply overwrites data. When this occurs, some messages are lost.

10.1.1 Printing the System Message Log File

To review the messages that have been logged, you should regularly use one of the following command sequences to print the /usr/adm/messages file.

To print the entire message log at your terminal, type this command sequence:

```
cat /usr/adm/messages
```

To print the entire message log at the line printer, type this command sequence:

```
lpr /usr/adm/messages
```

To print just the last 100 lines of the message log at your terminal, type this command sequence:

```
tail -100 /usr/adm/messages
```

To print the last 100 lines of the message log at the line printer, type this command pipeline:

```
tail -100 /usr/adm/messages | lpr
```

NOTE

Because the system usually crashes after a panic, the dmesg program is not running. Although the system writes panic messages to the message buffer, they are not copied to the system message log. To print the panic messages that were written to the message log just before a system crash, use the cda program. For further information, read Section 9.5, Using the cda Program.

10.2 Panic Messages

When it detects an internal error that it can neither correct nor ignore, the ULTRIX-11 kernel enters panic state. During a panic state, the kernel calls its panic() routine to perform these housekeeping tasks:

1. Calls the update() function. This system function writes out to disk (flushes) all file system changes that currently are in memory and cleans up all buffered I/O. This is equivalent to executing a sync command.

2. Calls the kernel printf() function. This system function prints an error message at the system console in the format:

   panic: message

   This message is a cryptic indicator of what caused the crash.

3. Calls the idle() function. This system function lowers the processor priority to zero and loops on a wait instruction. This lets the system service interrupts from any in-progress I/O and lets the system clock continue operating. The latter makes the switch register print function available for troubleshooting purposes. For further information, read Section 9.1, Console Switch Register Display Function.

A panic state can occur during either system startup or system operation. The most common causes are:

- I/O errors
- Overloaded file systems
- Improperly specified root and swap devices
- Corrupted kernel memory
- Bad memory location (nonparity memory)
- Improperly modified software
- Hardware problems

Because user processes cannot write to the ULTRIX-11 kernel, a bad memory location indicates that the corrupted kernel memory could only have been caused by a write from within the ULTRIX-11 kernel itself, by a stray NPR, or by a memory management fault.

The next 26 sections discuss the kernel panic messages.

10.2.1 panic: alloc

This message indicates that the system encountered a buffer that contained the superblock information for a mounted file system but did not have the B_MOUNT flag set. When it

allocates a buffer to contain superblock information, the system sets the B_MOUNT flag and treats the buffer differently than it does those allocated for ordinary files. This panic normally is preceded by this message:

SUPERB not B_MOUNT! on dev MAJOR/MINOR

10.2.2 panic: blkdev

This message indicates that the system attempted to assign an I/O buffer to a bad logical device. This panic occurs when the major device number is not that of a block mode device. That is, it is not in the bdevsw table in the system configuration file, /sys/conf/c.c.

10.2.3 panic: buffers

This message indicates that the system has insufficient memory for the buffer pool. During initialization, the ULTRIX-11 kernel first sets up a map of free memory and then allocates the buffer pool from it.

10.2.4 panic: bunhash

This message indicates that the system could not find an entry in the hash table for a buffer. The system maintains an entry in its internal hash table for each buffer that currently is being used. This panic occurs when the system's internal bunhash() function attempts to free a buffer and cannot find its hash table entry.

10.2.5 panic: devtab

This message indicates that the system attempted to assign an I/O buffer to a bad logical device. This panic occurs when the device table entry in the bdevsw table for the major device is null. This usually indicates that the device is not configured into the system.

10.2.6 panic: iinit

This message indicates that the system could not mount the root file system because it encountered an unrecoverable I/O error while attempting to read the superblock. The most likely causes are:

- Logical block one of the root file system either has developed a bad spot or has been otherwise corrupted.
- The system disk hardware has malfunctioned. Because the ULTRIX-11 operating system is running and was booted from the root file system on the system disk, the fault most likely is intermittent.

- The ULTRIX-11 operating system is configured for the wrong system disk. For example, if an ULTRIX-11 kernel configured for RP02/3 disks is booted from an RK07 disk, this panic results.

Immediately after booting, the ULTRIX-11 operating system initializes itself. Mounting the root file system is one step in the system initialization process. When the system mounts a file system, it reads a copy of that file system's superblock into memory. The system then uses the information in this in-memory superblock every time it accesses that file system. A file system's superblock is always located in logical block number 1 of that file system. Since the ULTRIX-11 operating system places the root file system at the beginning of the system disk, the superblock for the root file system is located in physical block 1 of that disk.

10.2.7 panic: init died

This message indicates that the system's control initialization process, /etc/init, no longer is running. Because the ULTRIX-11 system requires this process during operations, the system crashes whenever /etc/init is unavailable. For further information, read init(8) in the ULTRIX-11 Programmer's Manual, Volume 1.

10.2.8 panic: IO err in swap

This message indicates that the system encountered an unrecoverable error while either writing a process image out to or reading a process image in from the swap area. This panic can be caused by a bad block in the swap area, by faulty disk hardware, or by a memory parity error (swapped image).

10.2.9 panic: MSCP cntlr # fatal error:

This message indicates that, during normal operations, the system encountered an unrecoverable error on the specified controller (0, 1, or 2). In addition, the kernel prints this information:

```
CSR=# SA=# state=#
```

The CSR number indicates the controller's CSR address. The SA number indicates the contents of the controller's SA register. To indicate a fatal error, bit 15 equals 1 and bits 0 through 10 are the fatal error code. The fatal error codes are:

01 - Envelope/Packet Read (parity or timeout)
02 - Envelope/Packet Write (parity or timeout)
03 - Controller ROM and RAM parity
04 - Controller RAM parity
05 - Controller ROM parity
06 - Ring Read (parity or timeout)
07 - Ring Write (parity or timeout)
10 - Interrupt Master
11 - Host Access Timeout (higher-level protocol-dependent)
12 - Credit Limit Exceeded
13 - Unibus Master Error
14 - Diagnostic Controller Fatal Error
15 - Instruction Loop Timeout
16 - Invalid Connection Identifier
17 - Interrupt Write Error
20 - MAINTENANCE READ/WRITE Invalid Region Identifier
21 - MAINTENANCE WRITE Load to non-loadable controller
22 - Controller RAM error (non-parity)
23 - INIT sequence error
24 - High-level protocol incompatibility error
25 - Purge/poll hardware failure
26 - Mapping Register read error (parity or timeout)
27 - Attempt to set port data transfer mapping when option not present
** - Remaining values are unassigned

The state number indicates the current state of the controller:

0 - Controller not yet initialized (IDLE)
1 - Initialization Step 1 (in progress)
2 - Initialization Step 2 (in progress)
3 - Initialization Step 3 (in progress)
4 - Set controller characteristics (in progress)
5 - Controller RUN state

NOTE

During a boot, the system can also print this as a warning message if a fatal error occurred while trying to initialize a nonsystem disk controller.

10.2.10 panic: no clock

This message indicates that the system could not determine which clock device is used. During initialization, the ULTRIX-11 kernel normally determines which clock is available for use. The system first tests for the presence of the KW11-L line clock at address 0777546. If it responds,

the KW11-L is used. If the KW11-L does not respond, the system tests for the KW11-P programmable clock at address 0772540. If it responds, the KW11-P clock is used. If neither clock responds, this panic occurs.

10.2.11 panic: no imt

This message indicates that the system could not find an entry in the system mount table for a file system. This message is very similar to the no fs warning message. In this case, however, the system was attempting to establish the logical connection between the inode for the root directory of the mounted file system and the inode of the directory on which that file system is mounted.

10.2.12 panic: no procs

This message indicates that the system attempted to create a new process and could not locate an empty slot in the system process table. The ULTRIX-11 kernel manages processes using a corresponding entry for each in the system process table. The most likely time for this condition to occur is during system initialization. When it does occur at this time, it indicates that the ULTRIX-11 kernel could not create the init process. Because the ULTRIX-11 kernel checks for an empty process table slot before creating a process, this panic should not happen after system initialization.

10.2.13 panic: Out of swap

This message indicates either that there was no swap space available for the process's argument list or that the swap map was exhausted when the system attempted to swap out a process. When it uses the exec() system call to overlay a process, the kernel requires allocating a maximum of 10 blocks in the swap area as temporary storage for the argument list. The kernel also uses the swap map in allocating blocks in the swap area.

10.2.14 panic: out of swap space

This message indicates that, when the system attempted to swap out a process, either the disk swap area was full or there was not enough contiguous space available. This panic also indicates that the system generation is incorrect for the number of processes running on the system. When the system is generated correctly, the ULTRIX-11 kernel allocates sufficient swap space to prevent this situation from occurring.

10.2.15 panic: parity

This message indicates that, while in kernel mode, the system encountered either a memory parity error, a UNIBUS

parity error, or a double bit error in ECC MOS memory. If the parity error (trap through location 114) occurs while the ULTRIX-11 operating system is in user mode, the system terminates the user process, and no panic occurs. When this panic does occur, however, the system also prints the contents of all available memory error and status registers. For the PDP-11/44, PDP-11/60, and PDP-11/70, the system prints the contents of the memory system error registers. For all processors, the system prints the memory parity control and status registers.

10.2.16 panic: psig

This message indicates that, when attempting to send a signal, the system encountered an invalid signal number. The kernel uses its internal psig() function to send a signal to a process. This panic occurs when the psig() function encounters a signal that has been set to 0 (invalid reference).

10.2.17 panic: psig action

This message indicates that, when attempting to send a signal, the system encountered a reference for a signal that is either being held or ignored. The kernel uses its internal psig() function to send a signal to a process.

10.2.18 panic: remque

This message indicates that, when attempting to swap a process out, the system could not find an entry for it in the run queue. The kernel maintains an entry in the run queue for each process that is to be run. When a process is swapped out, the kernel normally removes its entry from the run queue.

10.2.19 panic: setrun

This message indicates that, when attempting to start a process, the system encountered a process that was not sleeping, idle, or stopped. The kernel's internal setrun() function is used to restart sleeping, idle, or stopped processes. This error occurs when the system attempts to restart a process and encounters a process that is not in one of these states.

10.2.20 panic: sleep

This message indicates that the system either encountered an invalid address on the wait channel or attempted to put to sleep a process the currently is not running. The kernel uses its internal sleep() function to let other functions sleep (give up the processor) until a specified event occurs. When a kernel function sleeps, an entry on the wait

channel (wchan) specifies the address of the event on which it waits.

10.2.21 panic: Timeout table overflow

This message indicates that the system attempted to store information in the timeout table when it was full. The timeout() function arranges for other functions to be executed after a specified time delay. The timeout table stores the function to be executed and the time delay.

10.2.22 panic: trap

This message indicates that the system encountered either a hardware trap while in kernel mode (other than through location 114), a power failure, or programmed interrupt request (PIRQ) while in user mode. A memory parity error while in user mode causes the system to terminate the user process and enter a parity error record in the system error log file. All other traps that occur while in user mode cause the system to terminate the user process and write the process image out (core dump) to a core file. The panic: trap message has this format:

```
ka6 = #
aps = #
pc = # ps = #
ovno = #
trap type #
panic: trap
```

The cda program uses the ka6 and aps numbers to locate the error information which is saved on the kernel stack by a panic trap. The pc and ps indicate the updated program counter and the processor status word, respectively. The ovno number specifies which text overlay segment was active when the trap occurred. The trap type indicates the nature of the error. The trap types are:

TRAP TYPE	TRAP VECTOR	CAUSE
0	4	Bus error
1	10	Reserved/illegal instruction
2	14	Breakpoint trace instruction
3	20	Input/output trap instruction
4	24	Power fail
5	30	Unexpected emulator trap (EMT)
6	34	System call from kernel mode (trap instruction)
7	240	Programmed interrupt request
10	244	Floating point exception
11	250	Memory management trap

NOTE

If the trap occurred while the processor was in user mode, octal 20 is added to the trap type. Trap type 24 indicates a power fail trap while in user mode.

When a panic trap occurs, use the cda program to extract the error information saved on the kernel stack by the trap handler. For further information on how to retrieve this information, read Chapter 9, ULTRIX-11 Crash Dump Analysis Facility. You also can manually access the panic-trap-error stack frame using one of three methods:

- The aps number is a kernel data space virtual address pointing to the top of the panic-trap-error stack frame. If the processor has a display register (PDP-11/45, PDP-11/55 or PDP-11/70), you can access the stack frame using the switch register display function. For further information, read Section 9.1, Checking the Console Switch Display Register. Load the aps value into the console switches and set the display select switch to the "display register" position. The system displays the old ps value in the display lights. For further information, see Figure 10-1, Panic Trap Error Stack Frame.

- For processors allowing virtual address access from the console (PDP-11/45, PDP-11/55, and PDP-11/70), you can access the stack frame with the console switches. Set the address select switch to "kernel D" and the display select switch to "data paths". Then, set the aps value in the switches and press load address followed by examine. Again, the system displays the old ps value in the display lights. The stack frame expands downward, so you must examine the stack in reverse order.

- You can access the stack frame in console physical mode. This is necessary on processors other than the PDP-11/45, PDP-11/55, or PDP-11/70. The ka6 number is the contents of memory management, active page register six and is used to map to the kernel stack. To determine the physical address of the top of the panic trap error stack frame, shift the ka6 number left by six places (multiply it by octal 100). Then, subtract 140000 from the aps number, and add the result to the left-shifted ka6. Finally, load the resulting physical address into the switches, and load address/examine to display the old ps value.

To prevent multiple panic traps, the system cannot reenter

the ULTRIX-11 trap handler. If a panic trap occurs while another panic trap is being serviced, the system enters a tight loop and ignores subsequent traps. This prevents multiple traps from masking the actual cause of the first.

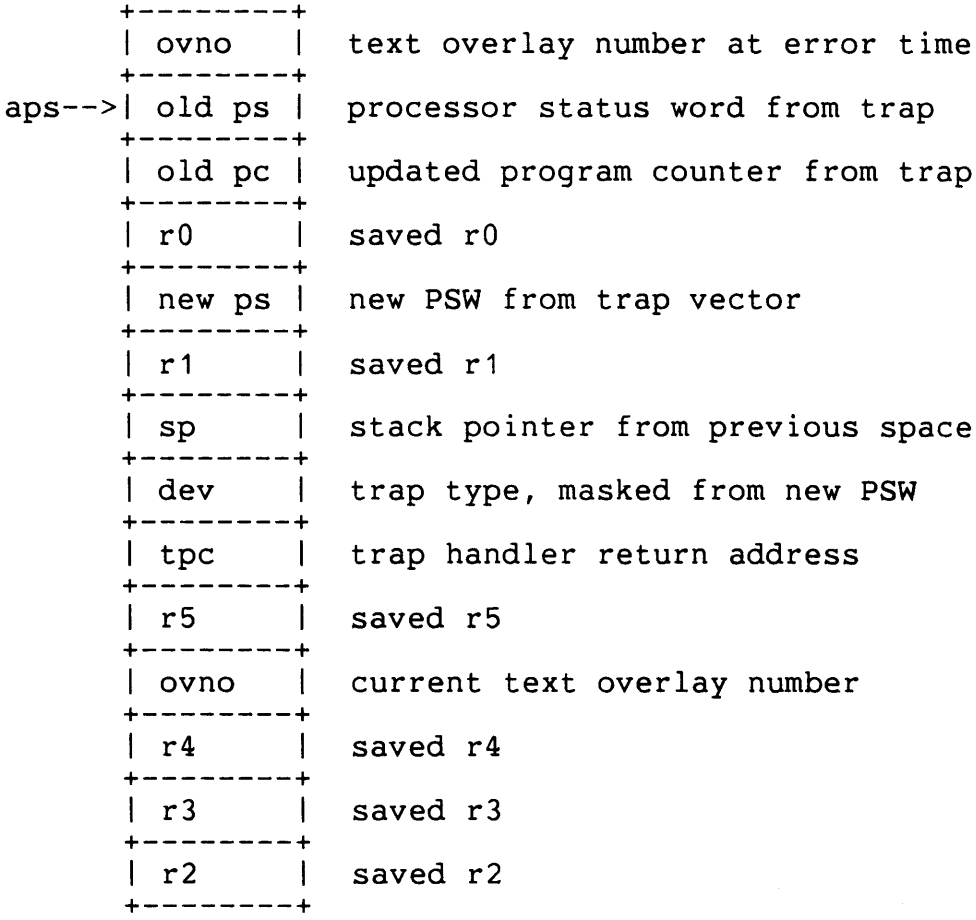


Figure 10-1 -- Panic Trap Error Stack Frame

10.2.23 panic: ttyrub

This message indicates that, when processing a terminal line editing function, the system encountered an invalid character class. During line editing, the kernel uses the character set table in determining the category of each ASCII character: ordinary, vertical tab, backspace, control, return, or tab. This panic occurs when the system encounters an entry in the character set table that does not belong to one of these categories.

10.2.24 panic: update

This message indicates that the system encountered an

invalid reference in the mount table. When a file system is mounted, the kernel allocates a buffer to contain its superblock information. Then, it creates an entry in the mount table that points to that buffer. This panic occurs when the pointer and the assigned buffer do not correspond.

10.2.25 panic: wakeup

This message indicates that the system encountered an entry in the sleep queue for a process that is neither sleeping nor stopped. The kernel maintains an entry in the sleep queue for each process that is sleeping or stopped. The kernel uses its wakeup() function to remove entries from the sleep queue. This panic occurs when the wakeup() function finds an entry whose process is not in the appropriate state.

10.2.26 panic: xfer size

This message indicates that, when attempting a UNIBUS transfer, the system encountered an invalid size. The UNIBUS map allocation routine normally limits the size of a data transfer to 57344 bytes. This panic occurs when the system attempts a transfer in excess of the 57344 bytes.

10.3 Warning Messages

System warning messages indicate that a nonfatal error has occurred. Although an error has occurred, it was not serious enough to warrant halting system operations. Because a warning message indicates that something is wrong and often precedes a fatal error, DIGITAL recommends that you investigate all warning messages.

The next sixteen sections discuss the system warning messages.

10.3.1 bad block on dev MAJOR/MINOR

This message indicates that the system encountered a bad block number when allocating or deallocating a data block in the file system located on the specified logical device (MAJOR/MINOR). The block is outside the defined range of the file system. The block either is part of the ILIST or is beyond the end of the file system. The system determines bad block numbers by comparing the block number to the isize and fsize values in the file system's superblock.

When this condition occurs, you should dismount the file system, check it with the fsck command, and repair it as necessary. For further information, read Section 1.2.17, Major/Minor Device Numbers, Section 4.1, File System Maintenance, and filsys(5) in the ULTRIX-11 Programmer's Manual, Volume 1.

10.3.2 bad count on dev MAJOR/MINOR

This message indicates that the system encountered an error when performing an internal consistency check of the in-memory free blocks and inodes. It also indicates that the file system located on the specified device (MAJOR/MINOR) has been corrupted. When a device number is mapped to the in-memory superblock, the system checks both the nfree and ninode counts. If they are greater than 100, the error results and both the nfree and ninode counts are zeroed. (This can also cause the system to print the no space warning.)

When this condition occurs, you should dismount the file system, check it with the fsck command, and repair it as necessary. For further information, read Section 1.2.17, Major/Minor Device Numbers, and Section 4.1, File System Maintenance.

10.3.3 Bad free count

This message indicates the same error condition as explained in Section 10.3.2, bad count on dev MAJOR/MINOR, except that it was detected at a different point in the operating

system.

When this condition occurs, you should dismount the file system, check it with the fsck command, and repair it as necessary. For further information, read Section 4.1, File System Maintenance.

10.3.4 core mapsize exceeded

This message indicates that the system has encountered a core (memory) map size overflow. The kernel maintains a map of the available free memory. When memory becomes too fragmented, the kernel may overflow this core map. When this occurs, you should shut down multiuser mode and halt the processor. Then, you should reboot your ULTRIX-11 system. If this condition persist, you should reconfigure your system and increase the map size.

10.3.5 err on dev MAJOR/MINOR

This message indicates that an error occurred on a block mode device. Because error logging was disabled, however, the system could not write the error information to the error log file. In multiuser mode, the ULTRIX-11 kernel captures information about block device I/O errors and writes the information in the error log. In single-user mode, error logging normally is disabled. In addition, the system prints this information:

```
bn=###### er=######,######
```

MAJOR/MINOR — Indicates the device on which the error occurred. For further information, read Section 1.2.17, Major/Minor Device Numbers, and Appendix C, ULTRIX-11 Device Names and Major Device Numbers.

bn — Indicates the logical block number relative to the start of the logical device. For block mode I/O, it is the block being transferred. For raw I/O, it is the block number where the transfer began. For further information about block and raw devices, read Section 1.2.15, Block I/O Mode, and Section 1.2.16, Character I/O Mode.

er — Indicates the octal contents of two of the device's hardware registers. For a list of the registers printed for each device, see Table 10-1, Registers Printed per Device. Because the driver issues a controller clear and does not log the error before initiating the retry, the system loses the contents of the remaining

device registers. The UDA50/KLESI/RUX1/RQDX1 driver prints MSCP error codes instead of hardware register contents. For further information, read Appendix F, UDA50/KLESI/RUX1/RQDX1 - MSCP Error Codes.

10.3.6 iaddress[#] > 2^24(#), i number = #, i dev =

This message indicates that the system encountered an invalid address when it attempted to update a disk-resident inode with the information contained in the in-memory inode table.

The ULTRIX-11 operating system stores the information that defines a file in a disk-resident structure called an inode. A portion of the inode contains the disk addresses which the system uses, either directly or indirectly, to allocate disk blocks to the file. The inode also holds the times of the last file access and the last update. For further information, read Section 1.2.10, Inode, and filsys(5) in the ULTRIX-11 Programmer's Manual, Volume 1.

When a file is opened, the system loads a copy of its inode into the in-memory inode table. At various times, the system examines all in-memory inodes. If an inode's access or update flags are set, indicating changes were made to the file, the system writes that in-memory inode out to the disk. Whenever it updates an inode, the system also checks the listed disk addresses to ensure it contains only valid disk addresses. The iaddress > 2^24 warning message occurs if any disk addresses are found to be greater than 2^24 (16777216). That is, the disk addresses are greater than the 24-bit length allotted to the disk address entry in an inode. The values are:

iaddress[#]	Indicates the element number in the inode's address array that contains the bad address.
2^24(#)	Indicates the actual disk address contained in the address array element.
i_number = #	Indicates the inumber of the corrupted inode.
i_dev = #	Indicates the octal major/minor number of the device that contains the corrupted inode.

10.3.7 issig

This message indicates that the system attempted to take action on a signal that was to be held or ignored. The kernel's issig() function checks the state of signals that are to be processed. Unless a process is being traced, the kernel normally does not act on signals that are to be held or ignored.

10.3.8 Inode table overflow

This message indicates that the system inode table is temporarily full. The system, therefore, cannot open any more files.

For each open file, the system maintains a copy of the inode that describes it in the in-memory inode table. Although a full inode table is not a fatal system error, the user process attempting to open the file receives a fatal error return. It is not unusual for this message to occur occasionally on an extremely busy system.

The user process recovers from this error by retrying the open after system activity has subsided. If this message occurs frequently, the system may not be properly configured to support the current number of users.

10.3.9 no file

This message indicates that the system file table is temporarily full. The system, therefore, cannot open any more files.

For each open file, the system maintains an entry in the in-memory file table. Although this is not a fatal system error, the user process attempting to open the file receives a fatal error return. It is not unusual for this message to occur occasionally on an extremely busy system.

The user process recovers from this error by retrying the file open after system activity has subsided. If the no file message occurs frequently, the system may not be properly configured to support the current number of users.

10.3.10 no fs

This message indicates that the system could not find an in-memory pointer to a file system. This occurs either when the ULTRIX-11 kernel searches the system mount table but cannot locate the in-memory copy of the file system's superblock or when the PIPE device is not mounted. (This normally indicates that the file system is not mounted). When the file system is mounted, the mount table should contain a pointer to the buffer in the internal buffer pool where a copy of the file system's superblock is located.

10.3.11 no space on dev MAJOR/MINOR

This message indicates that the file system located on the specified logical device (MAJOR/MINOR) has no free blocks remaining. The file system either is full or has been corrupted.

In either case, you should dismount the file system, check it with the fsck command, and repair it as necessary. For further information, read Section 1.2.17, Major/Minor Device Numbers, and Section 4.1, File System Maintenance.

10.3.12 out of text

This message indicates that there were no text table entries available when the system attempted to create a sharable process.

The system uses the text table to manage the shared text segment (pure code) and the proc table to manage the data segment. If a process has a shared text segment, the entry p_textp in the proc table points to the process's entry in the text table. This message also indicates that the system is heavily loaded and is not properly configured for the current number of users.

10.3.13 Out of inodes on dev MAJOR/MINOR

This message indicates that the system encountered an error when attempting to allocate an inode (create a file) on a file system that has no free inodes available.

This message also indicates that the file system is full. Even though a file system has free blocks, it can still be full. The number of files is limited not by the number of blocks that are available but by the number of inodes in the ilist.

When this condition occurs, you should dismount the file system, check it with the fsck command, and repair it as necessary. You also can make additional inodes available by removing excess files or enlarging the file system. For further information, read Section 1.2.17, Major/Minor Device Numbers, and Section 4.1, File System Maintenance.

10.3.14 proc on q

This message indicates that the system found an existing entry on the run queue (runq) when it attempted to create one for that process.

10.3.15 swap mapsize exceeded

This message indicates that the system encountered a swap map overflow. The kernel maintains a map of the available swap space. When the swap space becomes fragmented, the kernel may overflow the swap map. When this occurs, you should shut down multiuser mode and halt the processor. Then, you should reboot your ULTRIX-11 system. If this condition persists, you should reconfigure your system and increase the map size.

10.3.16 ?? unit # Write Locked

This message indicates that the system attempted a write operation on a disk that is write protected.

?? unit # Write Locked

The ?? is the ULTRIX-11 operating system mnemonic indicating the type of disk. For a list of device name mnemonics, see the first field in Table 10-1, Registers Printed per Device. For MSCP disks (RA60/RA80/RA81, RD51/RD52/RX50, and RC25), ?? is replaced by the disk controller type (UDA50/KLESI/RUX1/RQDX1). The # indicates the physical unit number.

ULTRIX-11 Name	Generic Name	First Register	Second Register
hk	RK06/7	RK control & status 2	RK error
hj	RM02/3/5, RP04/5/6, ML11 (on third RH)	RP control & status 2	RP error 1
hm	RM02/3/5, RP04/5/6, ML11 (on second RH)	RP control & status 2	RP error 1
hp	RM02/3/5, RP04/5/6, ML11 (on first RH)	RP control & status 2	RP error 1
hs	RS03/4	RS control & status 2	RS error
ht	TM02/3	TM error	TM control & status 2
hx	RX02	RX command & status	RX error & status
ra	RA60, RA80/RA81	MSCP opcode + flags (see Appendix F)	MSCP returned status (see Appendix F)
rc	RC25,	MSCP opcode + flags (see Appendix F)	MSCP returned status (see Appendix F)
rd	RD51/RD52	MSCP opcode + flags (see Appendix F)	MSCP returned status (see Appendix F)
rk	RK05	RK error	RK drive status
rl	RL01/2	RL control & status or RL multipurpose	RL disk address RL disk address
rp	RP02/3	RP error	RP drive status
rx	RX50	MSCP opcode + flags (see Appendix F)	MSCP returned status (see Appendix F)
tm	TM11	TM error	TM command & status
ts	TS11, TU80,TSV05,TK25	TS subsystem status or TS subsystem status	XSTAT0->XSTAT3 & retry count XSTAT0

Table 10-1 -- Registers Printed per Device

10.4 Miscellaneous Errors

This section discusses errors that fall into neither the panic nor warning category. The next seven sections discuss the miscellaneous error conditions.

10.4.1 Boot and Stand-alone Program Errors

This condition indicates that either the boot or stand-alone program encountered an error during a system installation.

In addition to the secondary boot program (/boot), the system is distributed with a number of stand-alone programs that you use during an ULTRIX-11 system installation. The boot and stand-alone programs operate in conjunction with a modified, stand-alone, single-user version of the ULTRIX-11 kernel.

When they encounter errors, the stand-alone ULTRIX-11 kernel and the stand-alone programs print error messages. These error messages are intended to be self-explanatory and usually result from improper file specifications or typing errors. For further information, read Section 3.8, Boot Error Messages.

10.4.2 Character I/O Device Errors

This condition indicates that one of the following errors occurred on a communication line:

- Parity errors
- Data overrun errors
- Framing errors

The ULTRIX-11 device drivers for the DH11/DHU11/DHV11, DZ11/DZV11/DZQ11, and DL11 communications devices log errors on terminal lines both by maintaining a count of the errors on each line and by saving the last faulty character along with the accompanying error bits.

Because of line noise and other causes, these driver errors can occur frequently and in large numbers. This makes printing an error message or writing to the error log file impractical. You can print the error counts and last error characters saved by the device drivers by using the pstat command with the -t option specified.

The system prints the error count under the heading ERRCNT and the last error character under the heading LASTEC. The LASTEC entry contains a copy of the received character buffer register from the device and includes the error character as well as the accompanying error bits.

For a description of received character buffer register bit

assignments, read the user's guide for your communications device. To print the error count and last error character from a crash dump file, use the cda command. For example, type this command sequence:

```
$ cda -pt /unix /usr/crash/core
```

10.4.3 Jump to Zero and Vector Through Location Zero

This condition indicates that the system encountered a panic trap. The ULTRIX-11 operating system defines a jump to location zero as setting the program counter to zero (for example, by a jump instruction with a destination address of zero).

The occurrence of this error is very unlikely. For both the separate I and D space and text overlay versions of the ULTRIX-11 operating system, a jump to zero results in a break point trace. The break point trace causes a panic trap (type two) with a PC of two. The cda program recognizes this special case of the panic trap error and flags it as a jump to zero.

The vector through location zero message indicates that the system encountered either a trap or an interrupt with a vector address of zero. The processor takes its new PC from location zero and its new PS from location two. This is a more common error than the jump to zero and can be caused either by a device which interrupts without asserting its vector address or by other UNIBUS related problems.

For the separate I and D space version of the ULTRIX-11 kernel, a vector through zero is logged as a stray vector. For the overlay text version of the ULTRIX-11 operating system, a vector through zero results in a panic trap (type zero) with a PC of five. The cda program flags this special type of panic trap as a vector through location zero.

10.4.4 Looping in Locore in User Mode

This condition indicates that the system could not create the init process. This can be caused by an I/O error while attempting to read the /etc/init file.

During the final step of the ULTRIX-11 operating system startup, the kernel creates the init process. The start-up code creates the init process by copying a small boot program into location zero in user space and transferring control to it.

This boot program consists of an exec() system call which is followed by a branch self instruction. When successful, the exec() system call then is overlaid by the init process.

The init process begins executing at location zero in user space and creates a process for each enabled terminal. If the exec() system call fails, a return occurs and the CPU hangs at the branch self instruction.

Because the clock interrupts and switches the processor to kernel mode periodically, it may not be apparent that the processor is looping in user mode at location six. If the processor is looping for at least some percentage of the time, halt the processor and disable the clock by resetting its interrupt enable bit to zero. If you are using the KW11-L clock, deposit a 0 in clock CSR 777546. If you are using the KW11-P clock, deposit a 0 in clock CSR 712540. Then press continue. If this error causes the looping condition, the address should be constantly at location six.

10.4.5 Red Zone Stack Violation

This condition indicates that the kernel stack has gone below the 16-word yellow zone. When a red zone fatal stack violation occurs, the CPU:

- Aborts current instruction
- Sets kernel stack pointer to 4
- Pushes PS and PC onto stack (locations 2 and 0)
- Traps through location 4

As a result of this red zone trap, the kernel stack pointer is zero. Like a yellow zone error, this error cannot result from an actual stack overflow. Instead it can be caused by modification of the kernel stack pointer or a detection logic fault. Unlike the yellow zone error, the red zone stack violation can be caused by the occurrence of one of the following hardware errors:

- Memory management aborts
- Nonexistent memory
- Odd address error
- UNIBUS parity error
- UNIBUS timeout

Because the stack pointer is set to zero on a red zone error, the ULTRIX-11 panic mechanism cannot handle this error. When this error occurs, the ULTRIX-11 kernel also prints this message and then halts:

```
RED ZONE
```

10.4.6 Stray Vector and Stray Interrupt

This condition indicates that the system encountered unexpected vectors through locations in the locore vector area. When this occurs, this information is printed:

SI # or SV #

The # indicates either the controller CSR address for a stray interrupt (SI) or the vector address for a stray vector (SV). When these errors occur, the system either crashes or, if it loses an interrupt, hangs. For further information about stray interrupts or vectors, read Section 8.4.7, Stray Interrupt Record, and Section 8.4.8, Stray Vector Record.

10.4.7 Yellow Zone Stack Violation

This condition indicates that the kernel stack has gone below the limit specified by the programmable stack limit register. That is, the kernel stack has fallen into the 16-word yellow zone (grace area) of the protected area below the stack. The ULTRIX-11 kernel always ensures that the memory management segment located immediately below the kernel stack is mapped for no access. This prevents the stack from overwriting system memory if a stack overflow occurs. Because the ULTRIX-11 kernel does not use the stack limit feature, the stack limit register will be zero (default limit octal 400). Before the detection of a yellow zone violation, a memory management access violation usually occurs.

10.5 User-Mode Errors

When an error occurs while a user process is running, the system prints a user-mode error message directly to the user's terminal.

If a hardware error occurs while the ULTRIX-11 operating system is executing a user process (processor is in user mode), the result normally is nonfatal. Because the effect of the error is usually confined to the user process only, it does not affect the operating system. If a power failure or programmed interrupt request occurs while the ULTRIX-11 operating system is in user mode, the results usually cause a system crash.

If a memory parity error occurs in user mode, the system enters a parity error record into the error log and terminates the user process that was running at that time. The user process is terminated without a core dump because writing the process's image out to disk would cause another parity error.

All other hardware traps cause the user process to be core dumped. A core dump involves printing an error message on the user's terminal and writing an image of the user process's memory out to a core file in the user's current directory.

To examine the core file and determine the cause of the core dump, use the adb program. For example, Figure 10.2 provides a sample adb session. In this example, a bus error occurred while executing the who command and adb was used to extract the error information. You can rename the core file, as shown, to prevent another core dump from overwriting it. The $C adb command prints a C stack backtrace. The $r prints the trap pc, ps, and general register contents. You can use the $f command to obtain the floating point status and error registers. The $q command exits the debugger. For further information, read adb(1) in the ULTRIX-11 Programmer's Manual, Volume 1.

```
$ who
Bus error - Core dumped
$ mv core who.core
$ adb /bin/who who.core
$C
~main(03,0177714)
        argc:   03
        argv:   0177714
$r
dev     020     (4) - Bus error
ps      0170000
pc      0210    ~main+0114
sp      0177672
r5      0177702
r4      0
r3      013254
r2      0
r1      061
r0      017302  _passwd
~main+0114:     beq     ~main+0124
$q
$
```

Figure 10.2 -- Sample adb(1) Session

Appendix A

Sysgen Program Example

This sample sysgen session illustrates:

- Logging in to the sys account
- Running the sysgen program
- Obtaining on-line help information
- Listing configurable processors and peripherals
- Creating a configuration file
- Making a new kernel
- Obtaining the kernel installation procedure
- Exiting the sysgen program

When creating the configuration file and making the new kernel, a sample response follows each prompt. If no response is given, the default is assumed. The help text for each prompt also is printed.

A-2 Sysgen Example

This example is of logging in to the sys account and running the sysgen program:

```
ULTRIX-11 System V2.0 (sysname)

login: sys
Password:

Welcome to the ULTRIX-11 System

$ cd conf
$ sysgen


ULTRIX-11 System Generation Program

For help, type h then press <RETURN>

sysgen>
```

This example is of obtaining on-line sysgen help information:

```
sysgen> h
```

The "sysgen>" prompt indicates the sysgen program is ready to accept commands. To execute a command you type the first letter of the command, then press <RETURN>. Some of the commands will ask you for additional information, such as a file name. For more help with a command, type h followed by the command letter then press <RETURN>. For example, "h c" for the create command.

Command	Description
<CTRL/D>	Exit from sysgen (backup one question in "c" command).
<CTRL/C>	Cancel current command, return to "sysgen>" prompt.
!command	Escape to the shell and execute "command".
[c]reate	Create an ULTRIX-11 kernel configuration file.
[r]emove	Remove an existing configuration file.
[l]ist	List names of all existing configuration files.
[p]rint	Print a configuration file.
[m]ake	Make the ULTRIX-11 kernel.
[i]nstall	Print instructions for installing the new kernel.
[d]evice	List configurable processors and peripherals.
[s]ource	Compile and archive a source code module (u1.c, etc.).

The sysgen sequence is: use "c" to create the configuration file, "m" to make the new kernel, and "i" for installation instructions.

A-4 Sysgen Example

This example is of listing the processors and peripherals that may be configured into the ULTRIX-11 kernel:

```
sysgen> d

Memory managed PDP-11 processors:

                (23, 23+, 24, 34, 40, 44, 45, 55, 60, 70, 73)

Disk Controllers:

Number  Name    Description
------  ----    -----------
1       hp      (first)  RH11/RH70 - 8 RM02/3/5, RP04/5/6, ML11
1       hm      (second) RH11/RH70 - 8 RM02/3/5, RP04/5/6, ML11
1       hj      (third)  RH11/RH70 - 8 RM02/3/5, RP04/5/6, ML11
1       hs      RJS04 (RH11/RH70) with up to 8 RS03/4
1       hk      RK611/RK711 with up to 8 RK06/7
1       ra      UDA50 with up to 4 RA80/RA81/RA60
1       rc      KLESI with up to 2 RC25 (4 units)
1       rd/rx   RQDX1 with up to 4 RD51/RD52/RX50
1       rx      RUX1 with up to 4 RX50
1       rp      RP11 with up to 8 RP02/3
1       rl      RL11 with up to 4 RL01/2
1       rk      RK11 with up to 8 RK05
1       hx      RX211 with one dual RX02 drive

Press <RETURN> for more:

Magtape Controllers:

Number  Name    Description
------  ----    -----------
1       ht      TM02/3 with up to 64 TU16/TE16/TU77
1       tm      TM11 with up to 8 TU10/TE10/TS03
1       ts      TS11/TSV05/TU80/TK25

Miscellaneous Devices:

Number  Name    Description
------  ----    -----------
1       lp      LP11 controller with 1 LP11 type printer
1       ct      C/A/T phototypesetter interface via DR11-C
```

```
Press <RETURN> for more:

Communications Devices:

Number  Name    Description
------  ----    -----------
8       dh      DH11 16 line asynchronous multiplexer
8       dhdm    DM11-BB modem control option for DH11
8       dhu     DHU11 16 line asynchronous multiplexer
4       dhv     DHV11 8 line asynchronous multiplexer
16      dz      DZ11 8 line asynchronous multiplexer
8       dzv     DZV11 4 line asynchronous multiplexer
8       dzq     DZQ11 4 line asynchronous multiplexer
16      kl      DL11/DLV11 single line unit (CSR 776500)
32      dl      DL11/DLV11 single line unit (CSR 775610)
4       du      DU11 single line synchronous interface
1       dn      DN11 4 line auto call unit interface
```

A-6 Sysgen Example

This example is of creating a sysgen configuration file:

```
sysgen> c

For help, answer the question with ?<RETURN>
To backup to the previous question, type <CTRL/D>

Configuration name <unix> ? ?


To use the default configuration name of "unix", press <RETURN>.

To use an alternate configuration name, enter the name and press
<RETURN>. The configuration name is limited to a maximum length
of eight characters. Digital recommends you use only alphanumeric
characters in the configuration name.

Configuration name <unix> ?

Configuration file exists, overwrite it <no> ? y

Processor type:

( 23 23+ 24 34 40 44 45 55 60 70 73 ) < 70 > ? ?


If the new kernel is being generated for the current CPU, press
<RETURN>. The number enclosed in < >, which is the current CPU
type, will be used. If the new kernel is for another system
enter the numeric portion of the processor type name, then press
<RETURN>. The numbers enclosed in ( ) list the supported CPU types.

For example, you would enter 70 for the PDP11/70 or 23+ for a
PDP11/23 plus processor.

The Micro/PDP-11 may be any of the following processor types:

        23+ - KDF11-B (F11)
        73  - KDJ11-A (J11)
        83  - KDJ11-B (J11)

If the target processor is not listed, select the processor type
that most closely resembles your processor. Remember, separate I
and D space is the most important processor feature.

Processor type:

( 23 23+ 24 34 40 44 45 55 60 70 73 ) < 70 > ?

Memory size in K bytes (K = 1024) < 1024 > ? ?
```

Sysgen is requesting the amount of memory on the target processor. The memory size is specified in K bytes, where, K is 1024 bytes. If the new kernel is being generated for the current CPU, press <RETURN> to use the value enclosed in < >, which is the current processor's memory size. If the new kernel is for another system, enter the memory size then press <RETURN>.

For example, if the processor has 256 K bytes of memory, you would enter 256, if the processor has one megabyte of memory you would enter 1024.

The minimum memory size is 192K bytes. The maximum memory size is 3840K bytes. 3840K bytes is four megabytes of memory minus the 256K bytes of address space reserved for the I/O page.

```
Memory size in K bytes (K = 1024) < 1024 > ?

I/O buffer cache size (NBUF: min = 16, max = 144) < 144 > ? ?
```

NBUF sets the size of the I/O buffer cache in the ULTRIX-11 kernel. Increasing the number of buffers should improve system performance. However, increasing NBUF also increases the amount of memory consumed by the operating system. Digital recommends you use the default NBUF for the initial system generation and delay experimenting with the size of the buffer cache until reliable system operation has been established.

To use the default value for NBUF, press <RETURN>. To change the size of the I/O buffer cache, enter the number of buffers then press <RETURN>.

Each NBUF costs 30 bytes of kernel data space for the buffer header and 512 bytes of memory (outside of kernel data space) for the actual buffer.

```
I/O buffer cache size (NBUF: min = 16, max = 144) < 144 > ?

Please enter the system (ROOT) disk controller first.

Disk controller type:

< rh11 rh70 rp11 rk611 rk711,
  rl11 rx211 rk11 uda50 rqdx1 klesi rjs04 rux1 > ? ?
```

Sysgen is requesting a list of all the disk controllers to be configured into the kernel. Enter the name of the system disk controller first, then enter the names of the other controllers on your system. When you have entered all your disk controllers, terminate the list by pressing <RETURN>. Consult the list below for the names and usage each type of disk controller.

When you enter a disk controller name, sysgen will ask a series

of questions about the controller and the drives connected to it. Type the answer to each question then press <RETURN>. Remember, you can just press <RETURN> to use the default answer or ?<RETURN> for help.

Note - all of the Q22 bus controllers may be used on processors with the 18 bit Q bus (jumper selectable). CAUTION, if a Q bus controller (rxv21, rlv11) is used on a processor with the Q22 bus, the disk may be accessed in buffered I/O mode only. Attempting RAW I/O transfers will cause errors. The PDP11/23 has an 18 bit Q bus, PDP11/23+ has a Q22 bus.

Press <RETURN> for more:

Name	Usage	Disk Drives Supported
rh11	Unibus	RM02, RP04/5/6, ML11
rh70	11/70 Massbus	RM02/3/5, RP04/5/6, ML11
rp11	Unibus	RP02/3
rk611	Unibus	RK06/7
rk711	Unibus	RK06/7
rl11	Unibus	RL01/2
rlv11	Q bus	RL01/2 (* specify rl11)
rlv12	Q22 bus	RL01/2 (* specify rl11)
rx211	Unibus	RX02
rxv21	Q bus	RX02 (* specify rx211)
rk11	Unibus	RK05
uda50	Unibus	RA60, RA80, RA81
rqdx1	Q22 bus	RX50, RD51, RD52
rux1	Unibus	RX50
klesi	Unibus/Q22 bus	RC25
rjs04	Unibus	RS03/4

```
Disk controller type:

< rh11 rh70 rp11 rk611 rk711,
  rl11 rx211 rk11 uda50 rqdx1 klesi rjs04 rux1 > ? rh70

First MASSBUS disk controller:

Drive 0 type < rm02 rm03 rm05 rp04 rp05 rp06 ml11 > ? ?
```

Sysgen is requesting a list of the drives connected to the disk controller. The names enclosed in < > are the drive types that may be attached to the specified disk controller. Enter the type of each drive, in order, starting with unit zero. To terminate the list of drive types, just press <RETURN>.

Sysgen assumes the disk units are numbered sequentially, starting with unit zero. To allow for non-sequential unit numbering, a drive type may be entered even if the disk drive is not physically present. The operating system will

ignore any non-existent units. For example, if three RP06 disks are to be numbered 0, 1, and 4, you would also specify drives two and three as RP06 disks. Drives two and three would be ignored by the system. Non-sequential unit numbering is not recommended, because it wastes kernel data space by allocating slots in the disk driver information tables for non- existent drives.

```
Drive 0 type < rm02 rm03 rm05 rp04 rp05 rp06 ml11 > ? rp06

Drive 1 type < rm02 rm03 rm05 rp04 rp05 rp06 ml11 > ? rm03

Drive 2 type < rm02 rm03 rm05 rp04 rp05 rp06 ml11 > ? rp06

Drive 3 type < rm02 rm03 rm05 rp04 rp05 rp06 ml11 > ?

CSR address <176700> ? ?
```

The number enclosed in < > is the default CSR address for the device. To use the default CSR address, press <RETURN>. If the device is not configured at the default CSR address, enter the actual address, then press <RETURN>. CSR addresses are always entered as octal numbers.

A device's CSR address specifies the I/O page address used by the operating system to access the device. The term CSR actually denotes the Control and Status Register, which is normally the first in a group of I/O page registers for the device.

```
CSR address <176700> ?

Vector address <254> ? ?
```

The number enclosed in < > is the default interrupt vector address for the device. To use the default vector, press <RETURN>. If the device is not configured at the default vector address, enter the actual vector address followed by <RETURN>. The vector address is always entered as an octal number.

A device's vector address is the address the processor will use to vector to the interrupt service routine for the device. The vector area is in low memory (locations 0 - 0776).

```
Vector address <254> ?

Is the system disk on this controller <yes> ? ?
```

Sysgen is asking if the system disk is connected to the current disk controller. The system disk is where the

ULTRIX-11 ROOT file system is located. Sysgen will ask this question only if the system disk has not already been specified.

If the system disk is on this controller, enter yes<RETURN> or just <RETURN>. If not, enter no<RETURN>.

```
Is the system disk on this controller <yes> ?

System disk unit number <0> ? ?
```

Sysgen is asking for the unit number of the system disk. The default unit number is <1> for the RC25 and <0> for all other disks. To use the default unit number, press <RETURN>.

You can specify a different unit number by entering the number followed by <RETURN>. If you do not use the default unit number, the following items must be considered:

- Not all hardware bootstraps can boot from units other than zero. You can load the boot from the distribution tape.
- The boot file specification will change. For example, unit two would be ??(2,0)unix, where ?? is the disk mnemonic.
- The /etc/fstab must be modified, see fstab(5) in the ULTRIX-11 Programmer's Manual, Volume 1.
- You must remake the file /dev/swap so that commands can access the swap area, see /dev/makefile.

```
System disk unit number <0> ?

Disk controller type:

< rh11 rh70 rp11 rk611 rk711,
  rl11 rx211 rk11 uda50 rqdx1 klesi rjs04 rux1 > ? rh70

Second MASSBUS disk controller:

Drive 0 type < rm02 rm03 rm05 rp04 rp05 rp06 ml11 > ? ml11

Drive 1 type < rm02 rm03 rm05 rp04 rp05 rp06 ml11 > ? rm03

Drive 2 type < rm02 rm03 rm05 rp04 rp05 rp06 ml11 > ?

CSR address <176400> ?

Vector address <204> ?

Disk controller type:

< rh11 rh70 rp11 rk611 rk711,
```

```
  rl11 rx211 rk11 uda50 rqdx1 klesi rjs04 rux1 > ? rl11

Drive 0 type < rl01 rl02 > ? rl02

Drive 1 type < rl01 rl02 > ? rl02

Drive 2 type < rl01 rl02 > ?

CSR address <174400> ?

Vector address <160> ?

Disk controller type:

< rh11 rh70 rp11 rk611 rk711,
  rl11 rx211 rk11 uda50 rqdx1 klesi rjs04 rux1 > ? uda50

First MSCP disk controller:

Drive 0 type < ra60 ra80 ra81 > ? ra60

Drive 1 type < ra60 ra80 ra81 > ? ra81

Drive 2 type < ra60 ra80 ra81 > ?

CSR address <172150> ?

Vector address <154> ?

Disk controller type:

< rh11 rh70 rp11 rk611 rk711,
  rl11 rx211 rk11 uda50 rqdx1 klesi rjs04 rux1 > ? rux1

Second MSCP disk controller:

Drive 0 type < rx50 > ? rx50

Drive 1 type < rx50 > ? rx50

Drive 2 type < rx50 > ?

CSR address <172150> ? 172144

Vector address <154> ? 150

Disk controller type:

< rh11 rh70 rp11 rk611 rk711,
  rl11 rx211 rk11 uda50 rqdx1 klesi rjs04 rux1 > ?

Use standard placement of root, swap, and error log <yes> ? ?
```

Sysgen contains tables that define the standard location of the ULTRIX-11 ROOT, PIPE, SWAP, and ERROR LOG file systems on each type of disk. Digital strongly recommends you use the standard placements for these file systems. Type yes<RETURN> or just <RETURN> to use the standard placements.

If you intend to experiment with the placement of these file systems, wait until the initial system installation has been completed and reliable system operation is established before generating a system with nonstandard placements. Also, backup you disks before booting a kernel with nonstandard placements!

To use nonstandard placements, answer no<RETURN>. Sysgen will ask a series of questions about the placement of the ROOT, SWAP, PIPE, and ERROR LOG file systems. Along with each question sysgen will print a default value, enclosed in < >, which is the standard placement for the item in question. You may use the default value or enter a new value. WARNING, sysgen accepts your answers without checking them for errors!

Press <RETURN> for more:

The following hints may be helpful:

- Placing the ROOT and SWAP on separate disk controllers will improve system performance. Placing them on different drives on the same controller is of little benefit.

- If the system make heavy use of pipes, placing the PIPE device on a separate disk controller should improve system performance. Otherwise PIPE and ROOT should be the same.

- All four file systems may exist within the same partition. Care must be taken to prevent file system overlap.

- The mkconf program (called by sysgen to make the kernel) does some checking of file system placements.

- Some of the auto-boot features may not function with non-standard placements of the ROOT, PIPE, SWAP, and ERROR LOG. Refer to Section 3 of the ULTRIX-11 System Management Guide.

- If the standard placements are not used, the only available crash dump devices will be magtape and RQDX1/RX50 (unit 2).

Use standard placement of root, swap, and error log <yes> ?

Magtape controller:

< tm02 tm03 tm11 ts11 tsv05 tu80 tk25 > ? ?

Sysgen is requesting a list of the magtape controllers to be configured into the new kernel. Most systems will have only a

single magtape controller, however, multiple controllers may be included. Only one of each type controller may be configured, that is, one TM02/3, one TM11, one TS11/TU80/TSV05/TK25.

Enter a magtape controller name, from the list below, then press <RETURN>. Sysgen will ask several questions about the controller and the drives connected to it. Answer each question then press <RETURN>. After you have entered the last magtape controller, terminate the list of controllers by pressing <RETURN>.

Name	Usage	Tape Drives Supported
tm02/3	Unibus	TU16, TE16, TU77
tm11	Unibus	TU10, TE10, TS03
ts11	Unibus	TS11
tsv05	Qbus/Q22bus	TSV05
tu80	Unibus	TU80
tk25	Qbus/Q22bus	TK25

```
Magtape controller:

< tm02 tm03 tm11 ts11 tsv05 tu80 tk25 > ? tm03

Number of magtape units <1> ? ?
```

Sysgen is requesting the number of tape drives connected to the magtape controller. Enter the number of magtape units then press <RETURN>. To use the default response of one unit, press <RETURN>.

Sysgen asks for the number of magtape units instead of the type of each unit, because the software drivers for magtapes adapt to the drive type automatically.

Sysgen expects magtape units to be numbered sequentially. However, non-sequential numbering may be used by setting the number of units to one more than the highest numbered unit. For example, if three tape units were to be numbered 0, 1, and 4, you would specify 5 magtape units. The system will ignore the nonexistent units. Non-sequential unit numbering is not recommended because the system must allocate space in the magtape driver information tables for nonexistent units.

```
Number of magtape units <1> ? 2

CSR address <172440> ?

Vector address <224> ?

Magtape controller:

< tm02 tm03 tm11 ts11 tsv05 tu80 tk25 > ?

Crash dump device < tm03 rp06 > ? ?
```

Sysgen is requesting the name of the crash dump device. Select the crash dump device from the list of names enclosed in < >. Enter the name then press <RETURN>. There is no default crash dump device, you must enter one of the names from the list.

The ULTRIX-11 system takes a crash dump by writing an image of memory to the crash dump device. The "memory image" is copied to a file on the system disk for analysis by the CDA (Crash Dump Analysis) program.

Digital recommends you select a magtape for the crash dump device if one is available. This will ensure that all of the system's memory will be saved in the crash dump.

If a magtape is not available, the crash dump can be written to the swap area of the system disk. If the system disk controller is an RQDX1, you may select the RX50 floppy disk drive as the crash dump device.

Depending on the type of disk and the amount of memory, the swap area may not be large enough to hold the entire "memory image". In this case, some crash dump data may be lost.

Crash dump device < tm03 rp06 > ? tm03

LP11 line printer present <no> ? y

CSR address <177514> ?

vector address <200> ?

Communications devices:

< dz dzv dzq dh dhu dhv dhdm du dn kl dl > ? ?

Enter the name of one of the communications devices listed below, then press <RETURN>. Sysgen will ask questions about the device. Answer these questions, then enter the name of the next device to be configured. If there are no more communications devices, press <RETURN> to terminate the list of devices.

```
Name  Device      Description
----  ------      -----------
dz    DZ11        8 line multiplexer
dzv   DZV11       4 line DZ11 for Q bus
dzq   DZQ11       4 line multiplexer (DZV11 replacement)
dh    DH11        16 line multiplexer
dhdm  DM11-BB     DH11 modem control
dhu   DHU11       16 line multiplexer
dhv   DHV11       8 line DHU11 for Q bus
du    DU11        synchronous line interface
dn    DN11        auto-call unit interface
kl    DL11/DLV11  (CSR 776500) single line unit
dl    DL11/DLV11  (CSR 775610) single line unit

Press <RETURN> for more:

The first "kl" is reserved for the console terminal. The console
terminal is automatically configured, do not count it the "kl"
specification. Use the "kl" and "dl" names for the equivalent
DLV11 Q bus devices.

Communications devices:

< dz dzv dzq dh dhu dhv dhdm du dn kl dl > ? dz

Number of units <1> ? 3

CSR address <160100> ? 160110

Vector address <300> ? 330

Communications devices:

< dz dzv dzq dh dhu dhv dhdm du dn kl dl > ? dh

Number of units <1> ?

CSR address <160020> ?

Vector address <300> ? 310

Communications devices:

< dz dzv dzq dh dhu dhv dhdm du dn kl dl > ? dhdm

Number of units <1> ?

CSR address <170500> ?

Vector address <300> ?

Communications devices:

< dz dzv dzq dh dhu dhv dhdm du dn kl dl > ?
```

```
Include C/A/T phototypesetter driver <no> ?

User devices:

< u1 u2 u3 u4 > ? ?
```

Sysgen allows you to configure up to four user written device drivers into the ULTRIX-11 kernel. If there are no user devices, press <RETURN>. Otherwise, enter the name of the first user device (u1, u2, u3, u4). Sysgen will ask a series of questions about the user device. Answer these questions, then enter the name of the next user device. If there are no more user devices, press <RETURN> to terminate the list.

To create a user written device driver, examine one of the user device driver prototype files (u1.c u2.c u3.c u4.c) in the /sys/dev directory. These files contain empty functions that define the interface to the ULTRIX-11 kernel. Edit your driver source code into these empty functions. Use the "s" command to compile and archive the new driver. Use the "m" command to make and install a new kernel.

For additional information, refer to Section 2.8 of the ULTRIX-11 System Management Guide.

```
User devices:

< u1 u2 u3 u4 > ?

Include Kernel floating point simulator <no> ? ?
```

If your processor is equipped with floating point hardware, type no<RETURN> or just <RETURN>. If your processor does not have the floating point hardware, type yes<RETURN>. Including the floating point simulation code will allow programs to execute floating point instructions on a processor without floating point hardware.

If the processor does not have floating point hardware and the floating point simulation code is not included in the kernel, any program that executes floating point instructions will be core dumped with an illegal instruction trap.

```
Include Kernel floating point simulator <no> ?

Use standard system parameters <yes> ? ?
```

The ULTRIX-11 system parameters specify the size of the kernel's internal data structures, such as the process table. The values of these parameters are used to adjust the sizes of the internal data structures to match the expected system load, that is, the number of users and job mix.

Sysgen contains a table of standard values for these parameters. Digital recommends that the standard parameters be used for the initial system generation, and that experimentation with the parameter values be delayed until reliable system operation is established. To use the standard parameters, answer yes<RETURN> or just <RETURN>.

To change the parameters, answer no<RETURN>. Sysgen will ask for the value of each parameter. Sysgen will also print the standard value of each parameter, enclosed in < >.

Press <RETURN> for more:

The system parameters are:

Param	OV_VAL	ID_VAL	COST	Description
NINODE	90	200	74	In-core inode table size
NFILE	80	175	8	Number of open files
NMOUNT	5	8	6	Number of mounted file systems
MAXUPRC	15	25	0	Maximum processes per user
NCALL	20	20	6	Number of callouts
NPROC	75	150	42	Number of processes allowed
NTEXT	25	40	12	Number of shared text segments
NCLIST	85	115	16	Number of cblocks in clist
CANBSIZ	256	256	1	TTY canon buffer size
NCARGS	5120	5120	0	Exec() argument list size
MSGBUFS	128	128	1	Error message buffer size
MAXSEG	61440	61440	0	Memory size limit
MAPSIZE	67	105	4	Core/swap map size

Use standard system parameters <yes> ? n

CHANGING SYSTEM PARAMETERS!

Press <RETURN> to use the default value!
Type ?<RETURN> for help!

ninode <200> ? ?

NINODE sets the size of the "in core inode" table in the ULTRIX-11 kernel. There will be an entry in this table for every open file, that is, device special file, current working directory, sticky text segment, open file, or mounted file system.

NINODE should be approximately NPROC+NMOUNT+(number of terminals). You can use the default value by pressing <RETURN>, or enter the value of NINODE followed by <RETURN>.

The cost of each NINODE is 74 bytes of kernel data space.

ninode <200> ?

```
nfile <175> ? ?
```

NFILE sets the size of the "open file" table in the ULTRIX-11 kernel. The size of this table limits the number of simultaneous open files the system may have.

NFILE should be about the same size as NINODE. To use the default value, press <RETURN>. To change NFILE, enter the new value then press <RETURN>.

The cost of each NFILE is 8 bytes of kernel data space.

```
nfile <175> ?

nmount <8> ? ?
```

NMOUNT sets the size of the mount table in the ULTRIX-11 kernel. The size of this table limits the number of mounted file systems to NMOUNT. Each mounted file system requires en entry in the mount table and a buffer, from the I/O buffer cache, to hold its superblock.

NMOUNT should be set to the number of permanently mounted file systems plus a number of temporary mounts. The permanent mounts can be determined by counting the active entries in the file system table (/etc/fstab). An active entry is one marked "rw" or "ro", not "xx". The number of temporary mounts depends on the system configuration and work load, two is generally enough.

Press <RETURN> to use the default NMOUNT, or enter the number of mounts followed by <RETURN>.

The cost of each NMOUNT is 6 bytes of kernel data space and the dynamic allocation of a buffer from the I/O buffer cache.

```
nmount <8> ?

maxuprc <25> ? ?
```

MAXUPRC sets the maximum number of processes that a user can have running simultaneously. MAXUPRC should be set just large enough that users can get work done but not so large that a user can consume all available processes, in the event of a programming error.

Press <RETURN> to use the default value, or enter an alternate value then press <RETURN>.

There is no data space cost associated with MAXUPRC.

```
maxuprc <25> ?
```

ncall <20> ? ?

NCALL sets the size of the callout table in the ULTRIX-11 kernel. Callouts are entered in this table when internal system timing must be done, such as carriage return delays for terminals.

The default NCALL size should be sufficient for most systems. To use the default value, press <RETURN>. To change the size of the callout table, enter the new value then press <RETURN>.

The cost of each NCALL is six bytes of kernel data space.

ncall <20> ?

nproc <150> ? ?

NPROC sets the size of the process table in the ULTRIX-11 kernel. The size of this table limits the number of processes that can be active in the system. Each active process requires an entry in the process table.

There is no set rule for the size of NPROC, it depends on how the system is being used. The default value should be sufficient for most systems, press <RETURN> to use the default NPROC. To change NPROC, enter the new value then press <RETURN>.

The cost of each NPROC is 42 bytes of kernel data space.

nproc <150> ?

ntext <40> ? ?

NTEXT sets the size of the "text" table in the ULTRIX-11 kernel. The size of the text table limits the number of shared text (pure code) segments that may be active in the system.

The default value for NTEXT should be sufficient for most systems, press <RETURN> to use the default value. NTEXT should be increased for systems with a large number of shared text processes. The NTEXT value can be changed by entering the new value followed by <RETURN>.

The cost of each NTEXT is 12 bytes of kernel data space.

ntext <40> ?

nclist <115> ? ?

NCLIST sets the number of 14 character clist segments (cblocks) allocated to the clist in the ULTRIX-11 kernel. Clists are used

to buffer characters for devices like terminals.

NCLIST should be large enough that the clists does not become exhausted at times of high terminal I/O activity. Enough clists should be allocated so that every terminal can have one average length line pending (about 30 or 40 characters).

The default NCLIST value should be adequate for most systems. To use the default value, press <RETURN>. To change the clist size, enter the new value then press <RETURN>. Use the following rule to calculate the number of Clists:

NCLIST = 55 + (3 times average number of active terminals)

The cost of each NCLIST is 16 bytes of kernel data space.

nclist <115> ?

canbsiz <256> ? ?

CANBSIZ specifies the size of the terminal canonicalization buffer in the kernel. This buffer is used for erase and kill processing when the system is accepting input from a terminal. That is, when you type <DELETE> to erase a character or <CTRL/U> to kill an entire line of input.

CANBSIZ limits the length of a terminal input line. The default of 256 should be large enough for most applications. The cost of each CANBSIZ is one byte.

canbsiz <256> ?

ncargs <5120> ? ?

NCARGS is the maximum number of characters allocated for the argument list when a process is created via the "exec" system call. NCARGS limits the number of arguments that can be passed to a process. Each "exec" system call requires (NCARGS+511)/512 contiguous blocks in the swap area, to hold the argument list.

The default NCARGS value should be large enough for most systems. To use the default value press <RETURN> or enter an alternate value followed by <RETURN>.

NCARGS use no kernel data space. However, setting NCARGS too high may cause swap space exhaustion or fragmentation.

ncargs <5120> ?

maxseg <61440> ? ?

MAXSEG limits the maximum amount of memory that the operating sysgen will use. To ensure that the system uses all available memory, use the default MAXSEG value by pressing <RETURN>. There is no harm is setting MAXSEG larger than the physical memory size.

MAXSEG is only changed for maintenance purposes, that is, to force swapping or avoid a known faulty section of memory. The values of MAXSEG is the number of 64 bytes memory segments to be used. The system will use all available memory up to and including the limit set by MAXSEG. For example, the default MAXSEG of 61440 allows the system to use up to 3.75 megabytes of memory (4Mb - I/O page). Setting MAXSEG to 16384 would limit the memory size to 1 megabyte.

```
maxseg <61440> ? 16384

msgbufs <128> ? ?
```

MSGBUFS specifies the size of the system error message buffer in the ULTRIX-11 kernel. All system error messages, printed on the console terminal, are also saved in this buffer for collection at a later time by the DMESG program. DMESG runs every 10 minutes and transfers the error messages from the kernel buffer to a file (/usr/adm/messages). If more than MSGBUFS characters of error message text are printed in a 10 minute period, some previous error messages will be overwritten. This is due to circular buffering.

Use the default MSGBUFS value by pressing <RETURN>, or enter an alternate value then press <RETURN>. The cost of each MSGBUFS is one byte of kernel data space.

```
msgbufs <128> ?

mapsize <105> ? ?
```

MAPSIZE sets the size of the core and swap maps in the ULTRIX-11 kernel. These maps are used for keeping track of free segments of memory and swap space. The default value is a function of the number of processes (NPROC); 30+(NPROC/2). The worst case value for MAPSIZE would be (2*NPROC)+2, though the maps rarely get to that size. MAPSIZE should only be changed if you get "mapsize exceeded" messages on the system console terminal; this is most likely on processors that have large memory sizes and do not have separate Instruction and Data space.

Press <RETURN> to use the default MAPSIZE or enter an alternate value then press <RETURN>.

The cost of each MAPSIZE is 4 bytes of kernel data space.

```
mapsize <105> ?
```

```
Line frequency in hertz <60> ? ?
```

Enter the AC power line frequency then press <RETURN>. The default value is 60 hertz. The line frequency should be 60 hertz for the United States or 50 hertz for Europe. If you insist, sysgen will accept any value for the power line frequency. This allows for the one hertz variation is AC line frequency that occurs in some areas.

```
Line frequency in hertz <60> ?

Timezone (hours ahead of GMT) <5=EST 6=CST 7=MST 8=PST> ? ?
```

Sysgen is requesting the timezone for your local area. Specify the timezone as the number of hours ahead of GMT (Greenwich Mean Time). Do not include daylight savings time in the timezone specification. For example, Eastern Standard Time is five hours ahead of GMT.

```
Timezone (hours ahead of GMT) <5=EST 6=CST 7=MST 8=PST> ? 5

Does your area use daylight savings time <yes> ? ?
```

The operating system automatically compensates for the presence or absence of daylight savings time. If your local area uses daylight savings time, answer yes<RETURN> or just <RETURN>. If not, answer no<RETURN>.

```
Does your area use daylight savings time <yes> ?
```

This example is of making an ULTRIX-11 kernel:

```
sysgen> m

Configuration name <unix> ?

****** CREATING ULTRIX-11 CONFIGURATION AND VECTOR TABLES ******

Device  Address Vector  units

console 177560   60
kw11-l  177456  100
kw11-p  172540  104
hm      176400  204     2
ra      172150  154     2
rq      172144  150     2
rl      174400  160     2
lp      177514  200     1
ht      172440  224     2       (crash dump device)
hp      176700  254     3
dhdm    170500  300     1
dh      160020  310     1
dz      160110  330     3

Filsys  Device  maj/min start   length

root    hp        9/0
pipe    hp        9/0
swap    hp        9/1   200     6000
errlog  hp       27/1   0       200

***** MAKING KERNEL FOR SEPARATE I & D SPACE PROCESSORS ******

as - -o l.o l.s
as -o dump_id.o mch0.s dump.s
as -o mch_id.o mch0.s mch.s
cc -c -O -DSEP_ID -DKERNEL c.c
cc -c -O -DSEP_ID -DKERNEL dds.c
cc -S -DKERNEL ec.c
ed - ec.s < :comm-to-bss
as - -o ec.o ec.s
rm ec.s

The output file will be named unix_id !!!!!

ovload

The unix_id sizes must be within the following limits:

root text segment > 49152 but <= 57344
overlay text segments <= 8192, 7 overlays maximum
bss + data segments <= 49088 total
```

```
root+(overlay 1, overlay 2,...overlay n)+data+bss=root+data=(total)

size unix_id
50688+(7808,7360,4608)+5080+39784=95552b=0272500b (70464 total text)
rm *.o

New kernel is now named 'unix.os'!

****** CHECKING SIZE OF NEW ULTRIX-11 OPERATING SYSTEM ******

'unix.os' within limits, SYSGEN successful!
```

This example is of obtaining the kernel installation procedure and exiting the sysgen program:

```
sysgen> i

Use the following procedure to install the new kernel:

  • Type <CTRL/D> to exit from the sysgen program.

  • Become superuser (type "su", then enter the root password).

  • Move the new kernel to the root (mv unix.os /nunix).

  • Type <CTRL/D> twice (to logout), then login to the operator
    account. Use the operator services "s" command to shutdown
    the system and "halt" command to halt the processor.

  • Use the manual boot procedure, described in section 3.4 of
    the ULTRIX-11 System Management Guide, to boot the new kernel.
    (For example, "Boot: rl(0,0)nunix" from an RL02 disk.

  • Save the old kernel then rename the new kernel "unix".
    (mv unix ounix; mv nunix unix; chmod 644 unix)

  • Set the date (date command), check the file systems (fsck),
    then type <CTRL/D> to enter multi-user mode.

sysgen> ^D      (Exit sysgen program)

$
```

Appendix B

Sysx Program Example

This sample sysx session illustrates:

- Running the sysx program
- Obtaining on-line help information
- Obtaining the exerciser module names
- Obtaining on-line help for the create command
- Creating an exerciser run script
- Printing the status of each exerciser module
- Exiting the sysx program

When creating an exerciser run script, a sample response follows each prompt. If no response is given, the default is assumed. The help text for each prompt also is printed.

B-2 Sysx Example

This example is of running the sysx program and obtaining help information:

```
# sysx

System exerciser control program
Type h for help

> h

The SYSX program has two modes of operation: "command mode"
and "run mode". SYSX is in run mode whenever exercisers
are running and in command mode when no exercisers are running.

The '>' prompt indicates that SYSX is in command mode while the
The following list of SYSX commands is broken down into two
groups; commands available in "command mode", and commands
available in "run mode".

Except for <CTRL/D> and <CTRL/C>, commands are
executed by typing the command letter followed by a <RETURN>.
The commands will ask for additional information, such as
script name, if required.  For more help type 'h' followed
by the command, 'h r' for help run.

Press <RETURN> for more:


Commands available in command mode:

<CTRL/D>        Exit from the sysx program.
<CTRL/C>        Cancel current command and return to the prompt.
! command       Execute an ULTRIX-11 command.
b               Backup, save an existing log file.
c               Create an exerciser run script.
d               Delete an exerciser run script.
l               Print log files on the terminal or line
                        printer.
n               Name the exerciser to run on each device.
p               Print the contents of a script.
r               Run an exerciser script.
s               Stop all exerciser(s).
x               Print a list of the existing exerciser run
                        scripts.

Press <RETURN> for more:


Commands available in run mode:
```

<CTRL/D>	Exit from the sysx program.
<CTRL/C>	Cancel current command and return to the prompt.
! command	Execute an ULTRIX-11 command.
l	Print log files on the terminal or line printer.
p	Print the status of the currently running script.
r	Restart system exerciser(s).
s	Stop system exerciser(s).

B-4 Sysx Example

This example is of obtaining the names of the sysx exerciser modules:

```
> n

        EXERCISER       DEVICES          COMMENTS

        cpx             CPU              All PDP11 processors
        fpx             FP11-A/B/C/E/F   PDP11/40 FIS not supported
                        FPF11            PDP11/23 & PDP11/24 Floating
                                         Point
        memx            MEMORY           All types of memory
        lpx             LP11             All LP11 type line printers
        cmx             DH11,            Communications devices
                        DHU11,DHV11,
                        DZ11,DZV11,DZQ11
                        DL11
        mtx             TM02/3,          TU16/TE16/TU77 magtapes
                        TM11,            TU10/TE10/TS03 magtapes
                        TS11             TS11/TSV05/TU80/TK25 magtapes
        hpx             RM02/3/5,        All disks on RH11/RH70
                        RP04/5/6,        controllers
                        ML11

Press <RETURN> for more:

        hkx             RK06/7           Disks on RK611/RK711
                                         controller
        rpx             RP02/3           Disks on RP11 controller
        rlx             RL01/2           Disks on RL11 controller
        rkx             RK03/5           Disks on RK11 controller
        rax             RA60/80/81       Disks on UDA50 controller
        rax             RD51/RD52/RX50   Disks on RQDX1 controller
        rax             RC25             Disks on KLESI controller
        hxx             RX02             Disks on RX211 controller
```

This example is of obtaining on-line help information about the sysx c command:

```
> h c


The 'c' command is used to create exerciser run scripts.
This may NOT be done while exercisers are running. The 'c'
command asks for the script name, which may be a combination
of letters and numbers up to 11 characters in length. If a
script by that name already exists, the 'c' command will ask
whether or not to overwrite the existing script.  A script
may consist of one or more exerciser names including, in
some cases, multiple copies of the same exerciser.

Next, 'c' prompts for an exerciser name. Respond with the
name of the exerciser to be entered into the script followed
by a <RETURN>. After the name is entered, 'c' will ask several
questions about the options for that exerciser. After each
question, the default answer <default> is printed, to use
the default answer type a <RETURN>. Answering the question
with a '?' will produce an expanded explanation of the
question. Typing <CTRL/D> will cause SYSX to cancel the
current entry and return to the exerciser name question.

Press <RETURN> for more:


After all of the option questions have been answered, 'c'
again prompts for an exerciser name. To enter another
exerciser into the script, type the name followed by a
<RETURN>. To end the script, respond to the exerciser name
question with just a <RETURN>. Refer to the 'p', 'r', and 's'
commands for information about printing, running, and
stopping the exerciser run script. Exercisers should be
entered into the script in the same order as they are
listed by the 'n' command.
```

B-6 Sysx Example

This example is of creating an exerciser run script:

```
> c

Script name <sysxr> ? test

Script exists, overwrite it <no> ? y

To cancel a script entry, type <CTRL/D> !
Answer any question with a '?' for help !

Exerciser name ? ?


Type the name of the exerciser for the device to be
exercised followed by a <RETURN>. An exerciser name may
be used more than once in the same script.  Use the 'n'
command to obtain the exerciser name for each device.

Exerciser name ? cpx

Output errors to log file <yes> ? ?


The error messages and other output from the exercisers may
be printed on the terminal or written out to a log file.
Either method is acceptable, however, if multiple exercisers
or multiple copies of the same exerciser are running, error
message output to the terminal could become scrambled.
Output should be to log files when multiple exercisers are
running ! The log file name is automatically generated by
the sysx program, use the 'p' command to obtain the log file
names. The 'l' command is used to print the contents of log
files.

Output errors to log file <yes> ?

Number of copies to run <1> ? ?


Some of the exercisers, such as CPX and FPX, allow multiple
copies of the same exerciser to be running concurrently.
This options allows the number of copies to be specified.
The number enclosed in < > is the recommended number of
copies, type a <RETURN> to use this default number. Otherwise,
type the number of copies to be run followed by a <RETURN>.
The maximum number of copies is 50. If the entire system is
to be exercised, i.e., all devices, memory, the CPU, and
floating point are running concurrently, it is strongly
recommended that only the default number of copies of CPX
and FPX be run. If CPX or FPX is the only exerciser running,
then up to 50 copies may be running concurrently.

Number of copies to run <1> ?
```

```
Exerciser name ? fpx

Output errors to log file <yes> ?

Data error printout limit <5> ? ?
```

For some devices, such as disks, a data mismatch error could result in several hundred lines of error printout. This parameter limits the number of data mismatch errors printed for each occurrence of a data error. For example: if an entire disk sector failed, 256 data mismatch error printouts, consisting of three lines each, would be generated. The default number of data mismatch errors to print is five and the maximum is 256.

```
Data error printout limit <5> ?

Drop device after how many errors <100> ? ?
```

If a device becomes completely inoperative or has a very high error rate, an inordinately large log file could result. In order to limit the size of the log file and to prevent choking the system with errors from a broken device, this parameter limits the number of errors on any given device. If this limit is exceeded, the exerciser for that device will be terminated. The default error limit is 100 and the maximum is 1000.

```
Drop device after how many errors <100> ?

Number of copies to run <2> ?

Exerciser name ? memx

Output errors to log file <yes> ?

Exerciser name ? lpx

Output errors to log file <yes> ?

LP continuous printing <no> ? ?
```

In order to save paper, the line printer exerciser (LPX) prints about 12 pages of actual printout then goes into a pause state. In the pause state lpx sends characters to the line printer but cancels the printout before it begins. This exercises the line printer controller and the system without wasting large amounts of paper. Answering yes to this question will cause the line printer to print continuously.

```
LP continuous printing <no> ?
```

```
LP (NO PRINT) exercise controller only <no> ? ?
```

In order to save paper, the line printer exerciser (LPX) prints about 12 pages of actual printout then goes into a pause state. In the pause state lpx sends characters to the line printer but cancels the printout before it begins. This exercises the line printer controller and the system without wasting large amounts of paper. Answering yes to this question will cause lpx to enter a constant pause state and never generate any actual printouts. This mode exercises the line printer controller and the system, but not the line printer itself.

```
LP (NO PRINT) exercise controller only <no> ?

LP pause (NO PRINT) time in minutes <15> ? ?
```

In order to save paper, the line printer exerciser (LPX) prints about 12 pages of actual printout then goes into a pause state. In the pause state lpx sends characters to the line printer but cancels the printout before it begins. This exercises the line printer controller and the system without wasting large amounts of paper. This parameter is used to specify the length of the pause time. To use the default time of 15 minutes type a <RETURN>, otherwise type the desired pause time followed by a <RETURN>. The maximum pause time is 480 minutes or 8 hours.

```
LP pause (NO PRINT) time in minutes <15> ?

Exerciser name ? cmx

Output errors to log file <yes> ?

Data error printout limit <5> ?

Drop device after how many errors <100> ?

For DL11, # is NOT unit number, type ? for help !
Device type & unit number
(dh#, dhu#, dhv#, dz#, dzv#, dzq#, dl#) ? ?
```

Each copy of the communications multiplexer exerciser (CMX) exercises one DH11, DHU11, DHV11, DZ11, DZV11, DZQ11 unit or from one to 16 DL11 units. Type the device type and the unit number in the following format; dh#, dhu#, dhv#, dz#, dzv#, dzq# or dl#, where # is the unit number except for the DL11. For the DL11 # is 0 for the first 16 DL11 units and 1 for the second 16 units. The actual DL11 unit or units to be exercised will be selected via the line select question asked later. DL11 unit 0 line 0, i.e., the first DL11 is

always the system console and cannot be exercised !

For example:

dh0	exercise the first DH11 unit
dhu1	exercise the second DHU11 unit
dzv0	exercise the first DZV11 unit
dl0	exercise some or all of the first 16 DL11 units

```
For DL11, # is NOT unit number, type ? for help !
Device type & unit number
(dh#, dhu#, dhv#, dz#, dzv#, dzq#, dl#) ? dh0

Use maintenance loopback mode <yes> ? ?
```

The DL11, DH11, DZ11, DZV11 and DZQ11 multiplexers have a maintenance loopback mode of operation, which loops the transmit leads of each port back to the input leads. When maintenance loopback mode is invoked, all lines on that unit are looped back. This is the normal method of exercising communications multiplexer lines. An alternate method is to connect a turnaround connector to each port to be exercised. This method is normally used if only a single line is to be exercised.

The DHU11 and DHV11 multiplexers also have maintenance loopback mode. However maintenance loopback mode is chosen for individual ports, allowing single lines to be tested using maintenance loopback mode without disturbing the other ports on the multiplexer.

NOTE: The second serial line unit, on the PDP11/24 processor, does not support maintenance loopback mode. The second SLU may be exercised as a DL11, however a turnaround connector must be used to loop the outputs back to the inputs.

```
Use maintenance loopback mode <yes> ?

Select line(s) <no> ? ?
```

The normal mode of operation for the communications multiplexer exerciser (CMX) is to exercise all lines on the selected DH11, DHU11, DHV11, DZ11, DZV11 or DZQ11 unit. However, one or more individual lines may be selected. This is done by typing the number of the line to be selected followed by a <RETURN>. The sysx program will continue to ask for line numbers until only a <RETURN> is typed.

The DL11 is a special case for line selection. The first 16 DL11 units are exercised as lines 0 through 15 of DL unit

zero, and the second 16 DL11 units are exercised as lines 0 through 15 on DL unit one. Line zero of the first DL11, i.e., the first physical DL11, is the system console and is never exercised. CMX will not allowed line zero of DL unit zero to be selected. For example, to exercise the second and third physical DL11 units select lines one and two of DL unit zero. To exercise physical DL11 unit 16, select line 0 of DL unit one.

```
Select line(s) <no> ?

Disable line(s) <no> ? ?
```

The normal mode of operation for the communications multiplexer exerciser (CMX) is to exercise all lines on the selected DL11, DH11, DHU11, DHV11, DZ11, DZV11 or DZQ11 unit. However, one or more individual lines may be disabled. This is done by typing the number of the line to be disabled followed by a <RETURN>. The sysx program will continue to ask for line numbers until only a <RETURN> is typed.

```
Disable line(s) <no> ? yes

Line ? ?
```

Type the number of the line to be deselected followed by a <RETURN>. If another line is to be deselected, type that line number followed by a <RETURN>. To terminate the line deselect process type just a <RETURN>. The number of lines per unit are:

DH11	16
DHU11	16
DHV11	8
DZ11	8
DZV11	4
DZQ11	4

```
Line ? 1

Line ?

Select bit rate for all lines <no> ? ?
```

The communications multiplexer exerciser (CMX) varies the bit rate between lines in a random fashion. It also varies the bit rate on each line periodically. Answering yes to this question will cause all lines to be exercised at the same constant bit rate. The sysx program will ask for the bit rate.

```
Select bit rate for all lines <no> ? yes

Bit rate <9600> ? ?
```

This parameter is used to set the transmit and receive speed on a line or group of lines. Type the desired speed followed by a <RETURN>. Available speeds are:

110	BPS
300	BPS
1200	BPS
2400	BPS
4800	BPS
9600	BPS

The bit rate need not be specified, to omit the bit rate specification type only a <RETURN>. In this case a random bit rate pattern will be used, unless a fixed bit rate has been specified for ALL lines.

NOTE: The DL11 does not support programmable bit rates. A fixed bit rate must be specified for each DL11 line.

```
Bit rate <9600> ?

How many minutes between I/O statistics printouts <30> ? ?
```

All of the disk exercisers and the communications exerciser generate periodic printouts containing the number of read and write operations, and the number of errors. This parameter may be used to specify the time in minutes between these printouts. The default time is 30 minutes. To use the default time type a <RETURN>, otherwise type the time interval in minutes followed by a <RETURN>. The maximum time is 720 minutes or 12 hours.

```
How many minutes between I/O statistics printouts <30> ?

Exerciser name ? mtx

Output errors to log file <yes> ?

Data error printout limit <5> ?

Drop device after how many errors <100> ?

Magtape controller type < ht, tm or ts - type ? for help ! > ? ?
```

Unlike the other exercisers, which run one copy of the exerciser for each unit to be exercised, the magtape exerciser (MTX) runs one copy for the controller and that

copy exercises all drives on the controller. This option specifies the type of magtape controller to be exercised. There is no default controller type. Type one of the following controller types followed by a <RETURN>:

```
        ht      for TM02/3 with TU16/TE16/TU77
        tm      for TM11 with TU10/TE10/TS03
        ts      for TS11/TSV05/TU80/TK25

Magtape controller type < ht, tm or ts - type ? for help ! > ? ht

Magtape unit number <all> ? ?
```

This option is used to select the magtape unit or units to be exercised. Type <RETURN> to exercise all available drives on the specified magtape controller. Otherwise, type the number of the unit to be exercised. The sysx program will continue to ask for magtape unit numbers until just a <RETURN> is typed. The maximum number of drives per controller are:

```
        ht      TM02/3                  64
        tm      TM11                     8
        ts      TS11/TSV05/TU80/TK25     1

Magtape unit number <all> ? 0

Magtape unit number <all> ?

Inhibit magtape unit status message <no> ? ?
```

The magtape exerciser (MTX) prints the type and status of each tape unit on the specified controller. Answering yes to this question will inhibit the unit status printout.

```
Inhibit magtape unit status message <no> ?

Suppress end of pass I/O statistics <no> ? ?
```

At the end of each pass, the magtape exerciser (MTX) prints the number of read and write operations, and the number of hard errors for each drive. Answering yes to this question will inhibit the end of pass printout.

```
Suppress end of pass I/O statistics <no> ?

Length of tape <500 feet> ? ?
```

This option specifies the length in feet of tape that will be used by the magtape exerciser (MTX). Typing just a <RETURN>

invokes the default length of 500 feet, otherwise type the number of feet to be used followed by a <RETURN>. The maximum length is 2400 feet and the minimum is 10 feet.

Length of tape <500 feet> ?

Exerciser name ? hpx

Output errors to log file <yes> ?

Data error printout limit <5> ?

Drop device after how many errors <100> ?

RH11/RH70 controller number < Type ? for help ! > ? ?

The (HPX) disk exerciser supports various combinations of RM02/3/5, RP04/5/6, and ML11 disks connected to up to three RH11 or RH70 controllers. The ULTRIX-11 RH controller number is not specified by the physical or electrical position of the RH controller on the bus. Respond to the question with one of the following controller numbers:

0 - The first RH controller with RM02/3/5, RP04/5/6 and/or ML11 disks connected to it. Disks referred to as "hp".

1 - The second RH controller with RM02/3/5, RP04/5/6 and/or ML11 disks connected to it. Disks referred to as "hm".

2 - The third RH controller with RM02/3/5, RP04/5/6 and/or ML11 disks connected to it. Disks referred to as "hj".

RH11/RH70 controller number < Type ? for help ! > ? 0

Unit number ? ?

Type the unit number of the disk to be exercised followed by a <RETURN>. There is no default unit number and only one unit number should be entered. Valid unit numbers are '0' through

Unit number ? 1

File system(s) <all> ? ?

Most disks are partitioned into 8 pseudo disks called file systems, refer to the "ULTRIX-11 System Management Guide" for an explanation of disk partitions. The default mode of operation is to exercise all file systems on the disk. To invoke the default mode type 'a', 'all', or <RETURN>. To exercise only one file system type the number of that file system followed by a <RETURN>, valid numbers are '0' through

```
File system(s) <all> ?

How many minutes between I/O statistics printouts <30> ?

Inhibit disk file system status printout <no> ? ?
```

The disk exercisers will not write on certain areas of the disk if those file systems are being used by the system software. A file system will be treated as read only if; it is the ROOT file system, i.e., where the ULTRIX-11 kernel resides, the swap area, the error log area, or if the file system is mounted. If there are read only file systems, the disk exerciser will print a list of these file systems and the reason that each file system was declared read only. Answering yes to this question will inhibit the printing of this list. In most cases it is wise NOT to inhibit the read only file system printout.

```
Inhibit disk file system status printout <no> ?

Exerciser name ? rax

Output errors to log file <yes> ?

Data error printout limit <5> ?

Drop device after how many errors <100> ?

MSCP disk controller number < Type ? for help ! > ? ?
```

The (RAX) disk exerciser supports multiple MSCP controllers on the same system. The MSCP controllers are:

```
        UDA50/UDA50A - for RA60/80/81 disks
        RQDX1 - for RD51/RD52/RX50 disks
        KLESI - for RC25 disks
        RUX1 - for RX50 disks
```

Controllers are assigned numbers based upon the order that they are specified during the system generation. Below is a listing of the MSCP controllers in the current configuration.

```
There are 2 MSCP controllers in the current system configuration.
        Controller #0 is a UDA50A
        Controller #1 is a RUX1

MSCP disk controller number < Type ? for help ! > ? 0

Unit number ? 0

Allow writes on customer data area <no> ? ?
```

The UDA50/RQDX1/KLESI disk exerciser (RAX) must be able to exercise fixed media disks without destroying data in the customer area. The fixed media disks are:

UDA50-RA60/RA80/RA81, RQDX1-RD51/RD52
and KLESI-RC25

Customer data is protected by allocating the last 32 blocks for RQDX1, the last 102 blocks for KLESI or the last 1000 blocks for UDA50 disks as a maintenance area.

Normally, the RAX program will read the entire disk but only write on the maintenance area. Answering yes to this question will cause the RAX program to enable writes to the customer area as well as the maintenance area of the disk.

Press <RETURN> for more:

CAUTION!, do not answer yes unless you are absolutely certain that it is safe to write on the customer area of the disk. Verify that the customer either has no data on the disk, or has preserved the data on another disk or on magtape.

Answering yes does not override the normal read only file system rules, that is, RAX will not write on the ULTRIX-11 ROOT, SWAP, ERROR LOG file systems or on any logically mounted USER file systems.

Allow writes on customer data area <no> ?

****** Writes will be to maintenance area only ******

File system(s) <all> ?

How many minutes between I/O statistics printouts <30> ?

Inhibit disk file system status printout <no> ?

Exerciser name ?

Confirm script complete <no> ? yes

B-16 Sysx Example

This example is of printing the status of each exerciser module and exiting the sysx program:

```
> p

Script name <sysxr> ? test


 # EXER  STATE      LOGFILE  OPTIONS
 1  cpx  stop    cpx_01.log
 2  fpx  stop    fpx_01.log  -n 5 -e 100
 3  fpx  stop    fpx_02.log  -n 5 -e 100
 4 memx  stop    memx_1.log
 5  lpx  stop     lpx_1.log  -p15
 6  cmx  stop   cmx_dh0.log  -i -n 5 -e 100 -dh0 -u 1 -b 9600
 7  mtx  stop    mtx_ht.log  -n 5 -e 100 -ht -d0 -f500
 8  hpx  stop   hpx_0_1.log  -n 5 -e 100 -c0 -d1
 9  rax  stop   rax_0_0.log  -n 5 -e 100 -c0 -d0

> ^D                    (Exits sysx program)
```

Appendix C

ULTRIX-11 Device Names and Major Device Numbers

The ULTRIX-11 software does not support all devices listed below. For a list of supported devices, read the ULTRIX-11 Software Product Description. The UDA50, KLESI, RUX1 and RQDX1 disk controllers share a common device driver and, therefore, the same major device number. Raw major device numbers 12 - 17 are dummy entries that ensure no block mode device has a raw major device number that falls within the range of valid block major device numbers.

Block Major	Raw Major	Name	Controller/Devices
0	18	rk	RK11 - RK05
1	19	rp	RP11 - RP02/3
2	20	ra	UDA50 - RA60/RA80/RA81
		rc	KLESI - RC25
		rd	RQDX1 - RD51/RD52
		rx	RQDX1/RUX1 - RX50
3	21	rl	RL11 - RL01/2
4	22	hx	RX211 - RX02
5	23	tm	TM11 - TU10/TE10/TS03
6	24		RESERVED (future use)
7	25	ts	TS11/TSV05/TU80/TK25
8	26	ht	RH11/RH70 - TM02/3 - TU16/TE16/TU77
9	27	hp	1st RH11/RH70 - RM02/3/5, RP04/5/6, ML11
10	28	hm	2nd RH11/RH70 - RM02/3/5, RP04/5/6, ML11
11	29	hj	3rd RH11/RH70 - RM02/3/5, RP04/5/6, ML11
12	30	hs	RH11/RH70 - RS03/4
13	31	hk	RK611 - RK06/7
14	32	u1	User Device 1
15	33	u2	User Device 2
16	34	u3	User Device 3
17	35	u4	User Device 4
	0	kl	DL11 (console terminal)
	1	ct	CAT (phototypesetter interface)
	2	lp	LP11
	3	dc	DC11
	4	dh	DH11
	5	dp	DP11
	6	uh	DHU11/DHV11
	7	dn	DN11
	8	dz	DZ11/DZV11/DZQ11
	9	du	DU11
	10	tty	(general tty interface)
	11	mem	(memory driver)
	12-17		DUMMIES (prevent overlap)
	36	dhdm	DM11-BB (DH11 modem control)
	37	kl	(CSR 776500) DL11/DLV11-A/B, DLV11-F/J
	38	dl	(CSR 775610) DL11-C/D/E, DLV11-E

Appendix D

Disk Logical Partition Sizes

Each diagram presented in this appendix illustrates the sizes of a nonpartitioned or partitioned disk unit. The size and uses of each partition are listed in this format:

n ###### usage

n Specifies the disk partition number.

Specifies the partition size.

usage Specifies how the partition can be used (for example, root, swap, or user file system).

The sizes table in the disk driver defines the size of the logical partitions in each disk unit.

Nonpartitioned Disks - RX50/RX02/RK05/ML11

RX50/RX02/RK05/ML11 disks use a single partition to cover the entire unit. The RX50, RX02, and RK05 disks usually are used for user file storage. The ML11 solid state disk, however, is used as either a swap device or a /tmp file system (system temporary files).

RX50

7	800 user

RX02 (single density)

500	user

RX02 (double density)

1001	user

RK05

0	4872	user

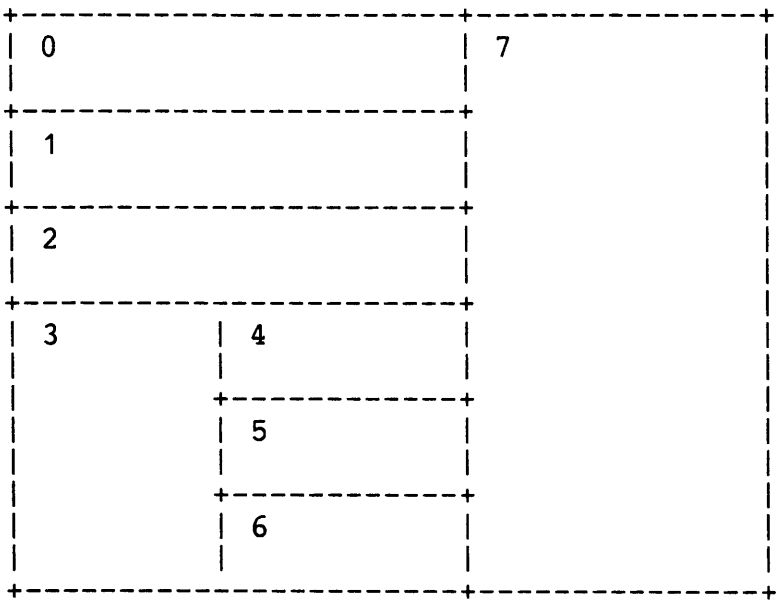
ML11

0	8192	user

Partitioned Disks

For partitioned disks, read the diagrams top-to-bottom and left-to-right. Partitioned disks may have one or more overlapping configurations, which use the same areas on disk. When you read the diagrams left-to-right, the overlapping partitions are mutually exclusive.

For example, in this diagram, partition 7 overlaps and excludes all others, while partions 4-6 overlap and exclude partition 3 only.



Partitioned Disks - RL01/RL02

If the RL01/2 is the system disk, all of RL02 unit 0 or RL01 units 0 and 1 are reserved for the ULTRIX-11 operating system. The remaining units are available for user file storage. If the system has only a single RL02 or two RL01 disks, user files can be stored in the root and /usr file systems. Space, however, will be limited.

If the RL01/2 is not the system disk, you can create user file systems on any unit. For the RL02 disk, you can use either partition 7 to create a single user file system or partitions 0 and 1 to create two smaller file systems. For the RL01 disk, you use either partition 0 or partition 7 to create a single user file system.

RL01 (unit 0)

0 8000 root 40 error log 2200 swap area	7 10240 user

RL01 (unit 1)

0 10240 /usr	7 10240 user

RL02

0 8000 root 40 error log 2200 swap area	7 20480 user
1 10240 /usr	

Partitioned Disks - RP02/3

If the RP02/3 is the system disk, unit 0 partitions 0, 1, and 2 are reserved for the ULTRIX-11 operating system. Partition 4 (RP02) and partition 3 (RP03) of unit 0 are available for user file storage. If the RP02/3 is not the system disk, you can create either a single or multiple user file systems on unit 0.

For RP02/3 units 1-7, you can use either the large partition to create a single user file system or the smaller partitions to create multiple user file systems. If you use partition 6 or 7, however, you cannot use partitions 0-4.

RP02

0 9600 root	6 40000 user
1 200 error log 5200 swap area	
2 10600 /usr	
4 14400 user	

RP03

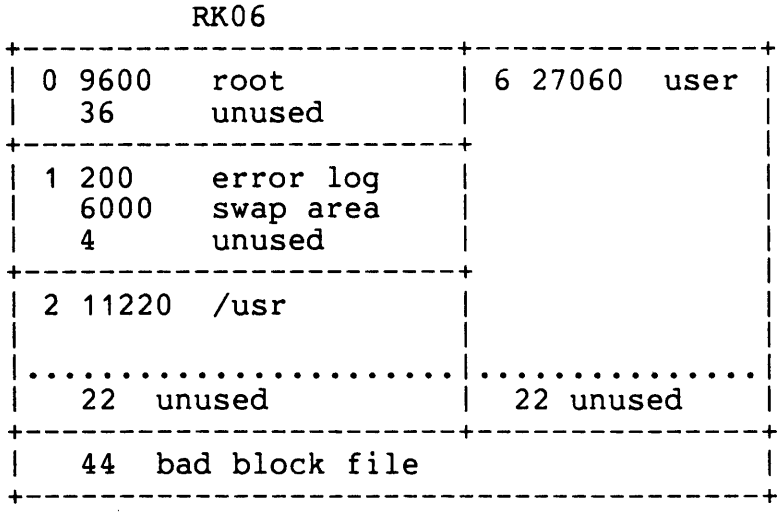
0 9600 root	7 80000 user
1 200 error log 5200 swap area	
2 10600 /usr	
3 54400 user	

Partitioned Disks - RK06

If the RK06 is the system disk, all of unit 0 is reserved for the ULTRIX-11 operating system. If the system has only a single RK06 disk, user files can be stored in the root and /usr file systems. Space, however, will be limited. If the RK06 is not the system disk, you can create either a single or multiple user file systems on unit 0.

For RK06 units 1-7, you can use either the large partition to create a single user file system or the smaller partitions to create multiple user file systems.

A bad block file, which can only be accessed by the operating system, resides at the end of the disk.

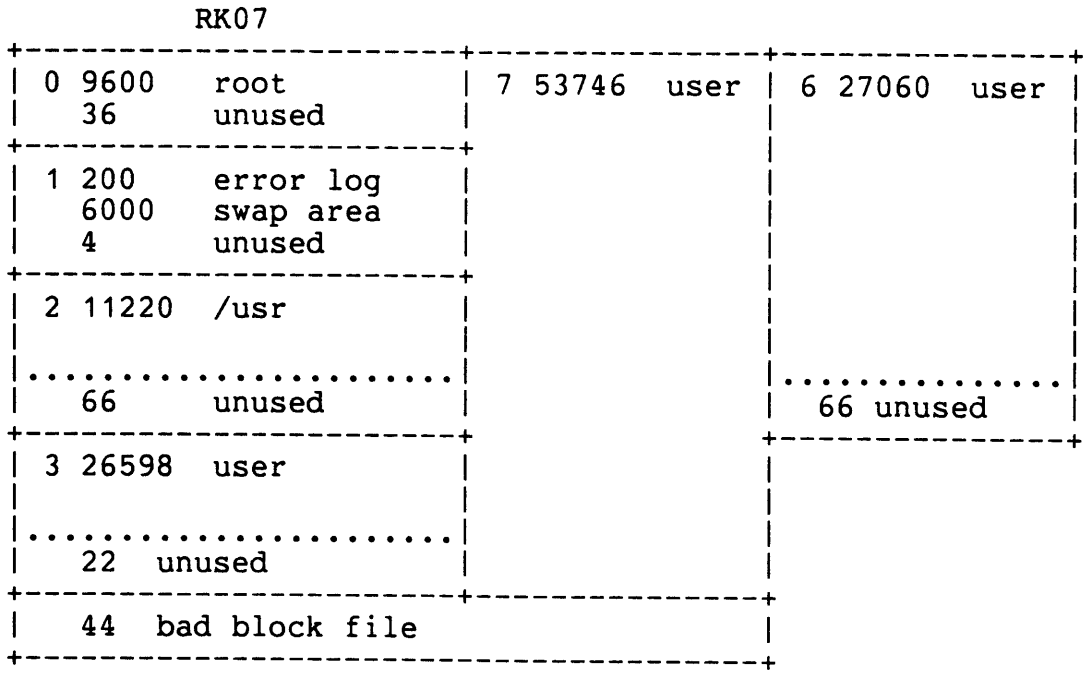


Partitioned Disks - RK07

If the RK07 is the system disk, unit 0 partitions 0, 1, and 2 are reserved for the ULTRIX-11 operating system. Partition 3 on RK07 unit 0 is available for user file storage. If the RK07 is not the system disk, you can create either a single or multiple user file systems on unit 0.

For RK07 units 1-7, you can use either the large partition to create a single user file system or the smaller partitions to create multiple user file systems. The RK07 partition 6 overlaps partitions 0-2 but not partition 3.

A bad block file, which can only be accessed by the operating system, resides at the end of the disk.

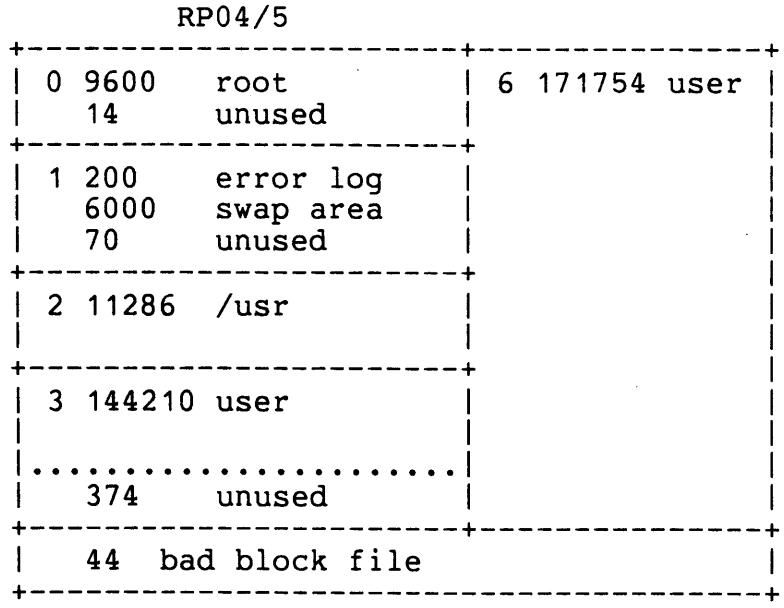


Partitioned Disks - RP04/5

If the RP04/5 is the system disk, unit 0 partitions 0, 1, and 2 are reserved for the ULTRIX-11 operating system. Partition 3 is available for user file storage. If the RP04/5 is not the system disk, you can create either a single or multiple user file systems on unit 0.

For RP04/5 units 1-7, you can use either the large partition to create a single user file system or the smaller partitions to create multiple user file systems.

A bad block file, which can only be accessed by the operating system, resides at the end of the disk.

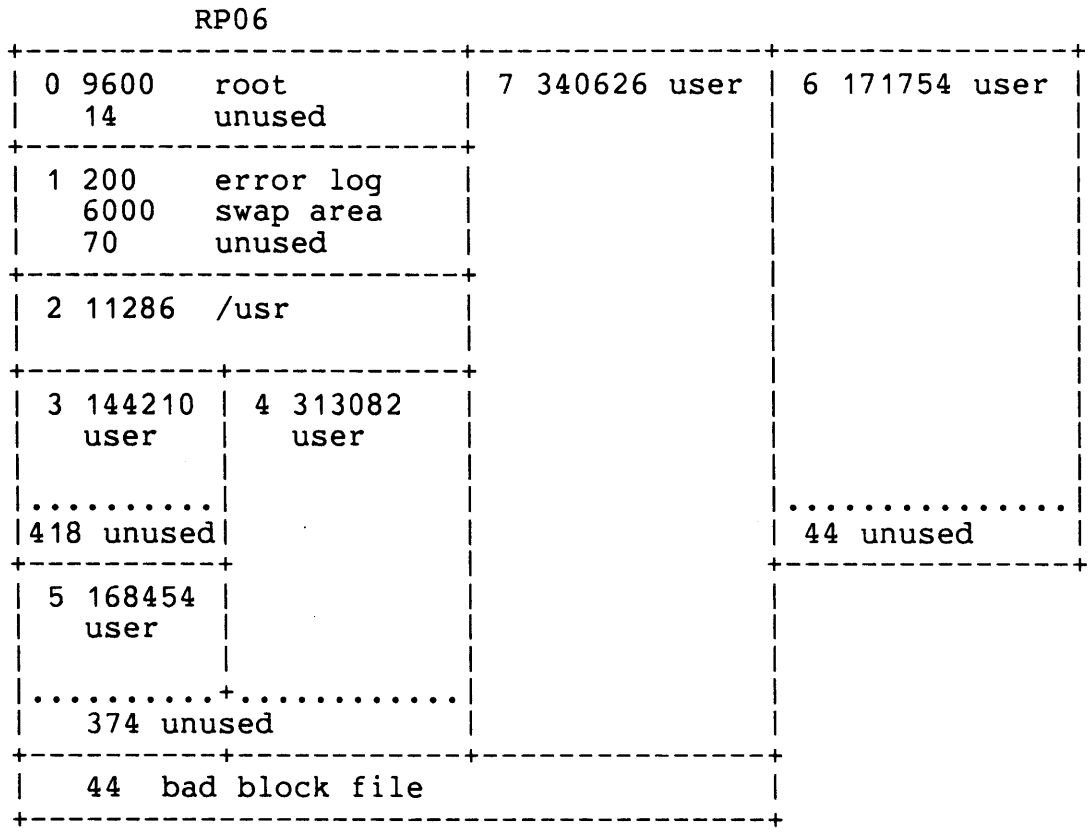


Partitioned Disks - RP06

If the RP06 is the system disk, unit 0 partitions 0, 1, and 2 are reserved for the ULTRIX-11 operating system. You can create user file systems either on partitions 3 and 5, or partition 4. If the RP06 is not the system disk, you can create either a single or multiple user file systems on unit 0.

For RP06 units 1-7, you can use either the large partition to create a single user file system or the smaller partitions to create multiple user file systems. DIGITAL recommends you do not use partition 6 of the RP06 disk.

A bad block file, which can only be accessed by the operating system, resides at the end of the disk.

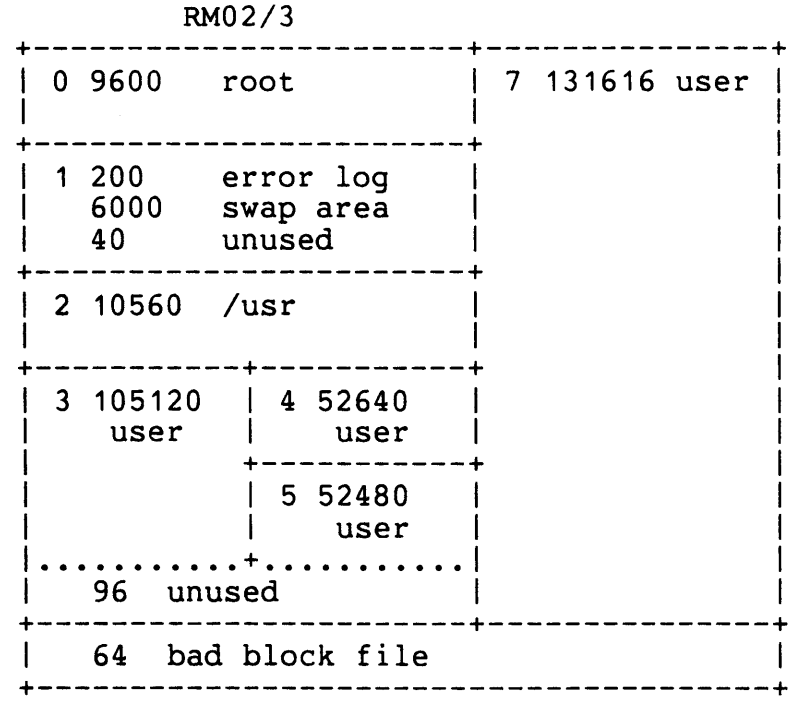


Partitioned Disks - RM02/3

If the RM02/3 is the system disk, unit 0 partitions 0, 1, and 2 are reserved for the ULTRIX-11 operating system. Partition 3 is available for user file storage. If the RM02/3 is not the system disk, you can create either a single or multiple user file systems on unit 0.

For RM02/3 units 1-7, you can use either the large partition to create a single user file system or the smaller partitions to create multiple user file systems.

A bad block file, which can only be accessed by the operating system, resides at the end of the disk.

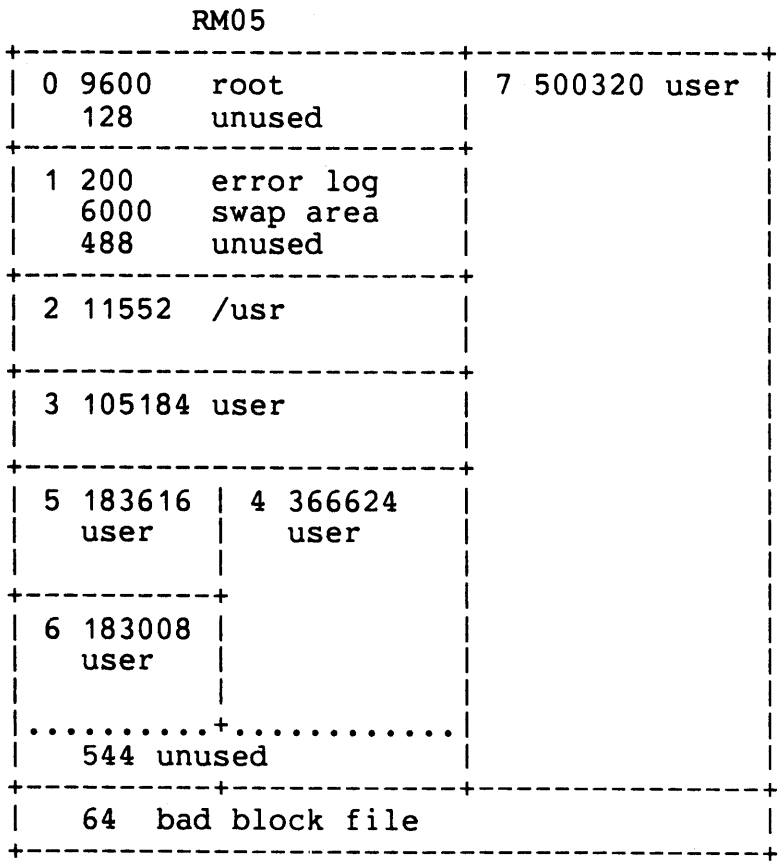


Partitioned Disks - RM05

If the RM05 is the system disk, unit 0 partitions 0, 1, and 2 are reserved for the ULTRIX-11 operating system. You can create user file systems on either partitions 5 and 6 or partition 4. Partition 3 is also available for user file storage. If the RM05 is not the system disk, you can create either a single or multiple user file systems on unit 0.

For RM05 units 1-7, you can use either the large partition to create a single user file system or the smaller partitions to create multiple user file systems.

A bad block file, which can only be accessed by the operating system, resides at the end of the disk.

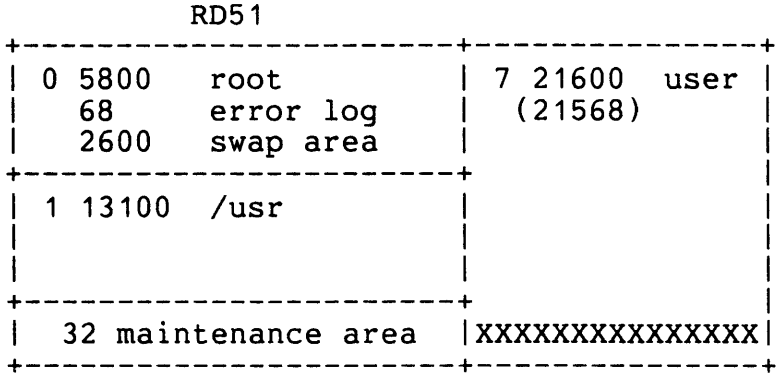


Partitioned Disks - RD51

If the RD51 is the system disk, unit 0 partitions 0 and 1 are reserved for the ULTRIX-11 operating system. If the RD51 is not the system disk, you can create either a single or multiple user file systems on unit 0. If the system has only a single RD51 disk, user files can be stored in the /usr file system.

If the system has multiple RD51 units, you can use either the large partition to create a single user file system or the smaller partitions to create multiple user file systems.

The RD51 disk has space at the end of the disk allocated for a maintenance area. This lets the USEP disk exerciser write on the fixed-media Winchester disks without destroying customer data. The area marked XXXXXXXXXXXXXXX indicates the overlap of partition 7 and the maintenance area. The number (21568) indicates the maximum size file system that you can create without overlapping the maintenance area on RD51 partition 7.

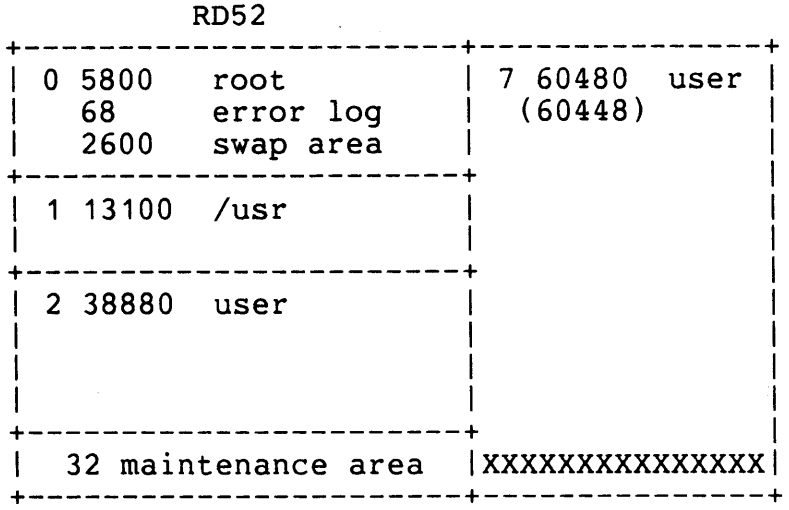


Partitioned Disks - RD52

If the RD52 is the system disk, unit 0 partitions 0 and 1 are reserved for the ULTRIX-11 operating system. Partition 2 is available for user file storage. If the RD52 is not the system disk, you can create either a single or multiple user file systems on unit 0.

For systems with multiple RD52 units, you can use either the large partition to create a single user file system or the smaller partitions to create multiple user file systems.

The RD52 disk has space at the end of the disk allocated for a maintenance area. This lets the USEP disk exerciser write on the fixed-media Winchester disks without destroying customer data. The area marked XXXXXXXXXXXXXXX indicates the overlap of partition 7 and the maintenance area. The number (60448) indicates the maximum size file system you can create without overlapping the maintenance area on RD52 partition 7.

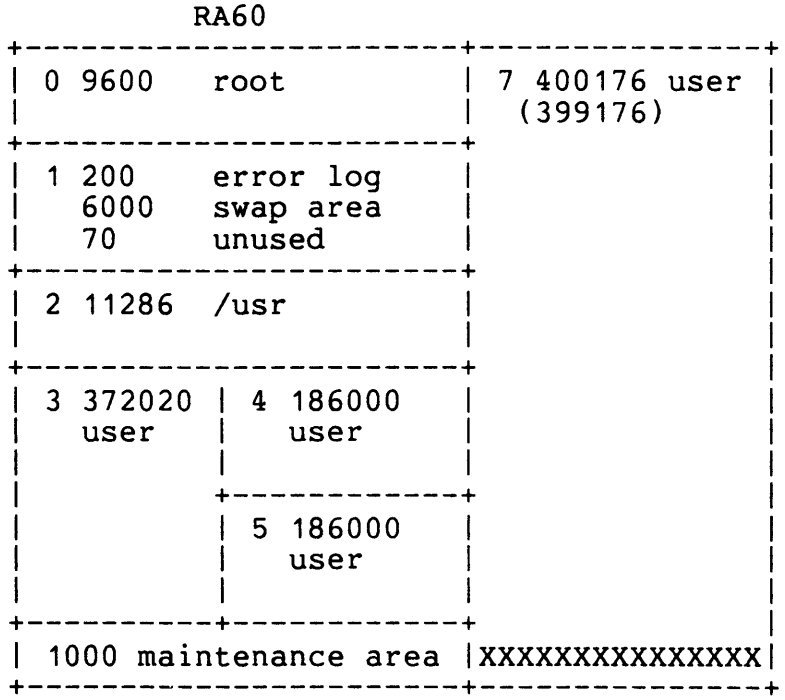


Partitioned Disks - RA60

If the RA60 is the system disk, unit 0 partitions 0, 1, and 2 are reserved for the ULTRIX-11 operating system. You can create user file systems on either partitions 4 and 5, or partition 3. If the RA60 is not the system disk, you can create either a single or multiple user file systems on unit 0.

For RA60 units 1-7, you can use either the large partition to create a single user file system or the smaller partitions to create multiple user file systems.

The RA60 disk has space at the end of the disk allocated for a maintenance area. This lets the USEP disk exerciser write on the disk without destroying customer data. The area marked XXXXXXXXXXXXXXX indicates the overlap of partition 7 and the maintenance area. The number (399176) indicates the maximum size file system you can create without overlapping the maintenance area on RA60 partition 7.

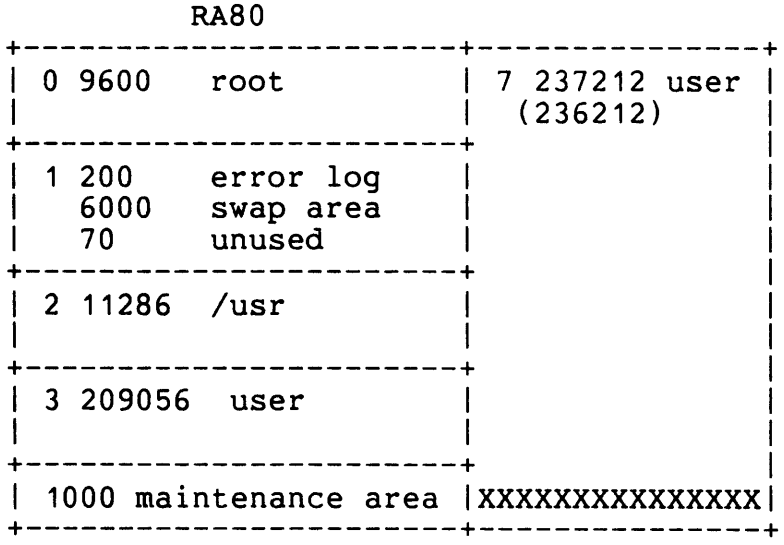


Partitioned Disks - RA80

If the RA80 is the system disk, unit 0 partitions 0, 1, and 2 are reserved for the ULTRIX-11 operating system. On unit 0, partition 3 is available for user file storage. If the RA80 is not the system disk, you can create either a single or multiple user file systems on unit 0.

For RA80 units 1-7, you can use either the large partition to create a single user file system or the smaller partitions to create multiple user file systems.

The RA80 disk has space at the end of the disk allocated for a maintenance area. This lets the USEP disk exerciser write on the fixed media Winchester disks without destroying customer data. The area marked XXXXXXXXXXXXXXX indicates the overlap of partition 7 and the maintenance area. The number (236212) indicates the maximum size file system you can create without overlapping the maintenance area on RA80 partition 7.

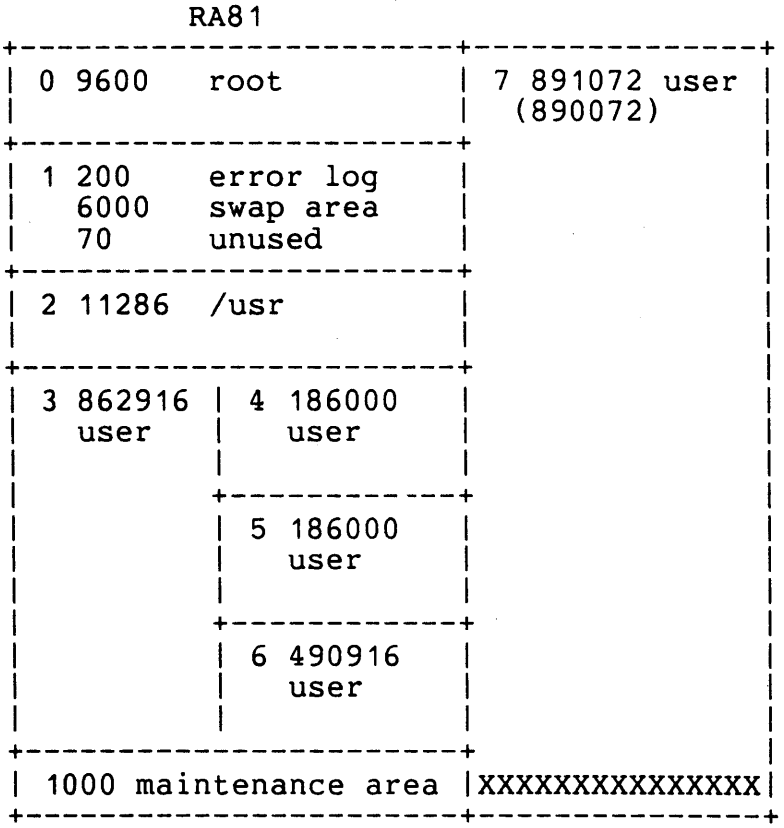


Partitioned Disks - RA81

If the RA81 is the system disk, unit 0 partitions 0, 1, and 2 are reserved for the ULTRIX-11 operating system. You can create user file systems on either partitions 4-6, or partition 3. If the RA81 is not the system disk, you can create either a single or multiple user file systems on unit 0.

For RA81 units 1-7, you can use the large partition to create a single user file system or the smaller partitions to create multiple user file systems.

The RA81 disk has space at the end of the disk allocated for a maintenance area. This lets the USEP disk exerciser write on the fixed media Winchester disks without destroying customer data. The area marked XXXXXXXXXXXXXXX indicates the overlap of partition 7 and the maintenance area. The number (890072) indicates the maximum size file system you can create without overlapping the maintenance area on RA81 partition 7.



Partitioned Disks - RC25

If the RC25 is the system disk, unit 1 partitions 0, 1, and 2 are reserved for the ULTRIX-11 operating system. On unit 1, partition 3 is available for user file storage. If the RC25 is not the system disk, you can create either a single or multiple user file systems on unit one.

For RC25 units 0, 2, and 3, you can use either the large partition to create a single user file system or the smaller partitions to create multiple user file systems.

The RC25 disk has space at the end of the disk allocated for a maintenance area. This lets the USEP disk exerciser write on the fixed media Winchester disks without destroying customer data. The area marked XXXXXXXXXXXXXXX indicates the overlap of partition 7 and the maintenance area. The number (50800) indicates the maximum size file system you can create without overlapping the maintenance area on RC25 partition 7.

RC25

0 9600 root	7 50902 user (50800)
1 200 error log 6000 swap area 70 unused	
2 11286 /usr	
3 23644 user	
102 maintenance area	XXXXXXXXXXXXXXX

Appendix E

Opser Program Example

This sample opser session illustrates:

- Logging in to the operator account
- Obtaining on-line help information
- Shutting down multiuser mode
- Checking for file system inconsistencies
- Restarting multiuser mode
- Exiting the opser program

E-2 Opser Example

This example is of logging in to the operator account and obtaining on-line opser help information:

```
login: operator
Password:

Welcome to the ULTRIX-11 System

ULTRIX-11 Operator Services

To correct typing mistakes:

        <DELETE> erases the last character,
        <CTRL/U> erases the entire line.

For help, type h followed by a return

opr> h

() - may use first letter in place of full name
Valid commands are:

!sh             - shell escape (execute ULTRIX-11 commands)
                  (Type <CTRL/D> to return from shell)
(u)sers         - show logged in users
(s)hutdown      - stop time-sharing
(f)sck          - file system checks
(r)estart       - restart time-sharing
(h)elp          - print this help message
backup cfn      - file system backup
                  (cfn = command file name)
halt            - halt processor
^D (<CTRL/D>)   - exit from opser

opr>
```

This example is of shutting down multiuser mode:

```
opr> s

ULTRIX-11 Shutdown
The following users are logged into the system

operator console Apr  6 10:05

How many minutes until shutdown [1-99] ? 2

Warning Phase

        1 minute warning sent
        2 minute warning sent
        FINAL WARNING SENT

Kill Process Phase

        Killing User Processes
        Killing System Processes
        Disabling Error Logging

Dismounting Mounted File Systems

Dismounting /dev/ml0    from /tmp
Dismounting /dev/hp02   from /sys
Dismounting /dev/hp03   from /usr
Dismounting /dev/hp05   from /staff

System Time-sharing Stopped

opr>
```

E-4 Opser Example

This example is of checking the root file system for inconsistencies:

```
opr> f

/dev/hp00
File System:  Volume:

** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Free List
636 files 7554 blocks 1660 free

^C      (file system check terminated <CTRL/C>)

opr>
```

This example is of restarting multiuser mode and exiting the opser program:

```
opr > r

Restarting ULTRIX-11 Time-sharing

Mounted /dev/ml0 on /tmp
Mounted /dev/hp02 on /sys
Mounted /dev/hp03 on /usr
Mounted /dev/hp05 on /staff

Enabling Error Logging

ERROR LOG has - 7 of 200 blocks used

Enabling terminals

Time-sharing Restarted

opr> ^D                 (Exits opser program)
Opser terminating

login:
```

Appendix F

UDA50/KLESI/RUX1/RQDX1 - MSCP Error Codes

The UDA50, KLESI, RUX1, and RQDX1 disk controllers report errors in the form of a Mass Storage Control Protocol (MSCP) error code. During normal system operation, the MSCP error code is entered into the system error log file. To decode and print the MSCP error code, use the error log print program, elp.

If for any reason the system cannot log errors, the MSCP error code is printed at the console terminal. When this occurs, you must manually decode it. Because system error logging is not enabled when the system is in single-user mode, this situation is most likely to occur during this time. In addition, the boot program and all of the stand-alone programs print the MSCP error code on the console terminal.

The MSCP error codes consist of up to three information fields: opcode/endcode, flags, status/event code.

The opcode identifies the type of command and the endcode indicates the response to it. The opcode and endcode usually are identical, except that bit 7 is set in the endcode. For example, the opcode for a read is 041 and its endcode is 0241.

The flags are contained in the high byte of an endcode. They report various side effects of an error (for example, bad block reporting).

The status code reports the completion status of an operation and the event code indicates the type of error. The status/event codes consist of a 5-bit code combined with an 11-bit sub-code.

F-2 MSCP Error Codes

OPCODE	ENDCODE	OPERATION TYPE
001	0201	ABORT Command
020	0220	ACCESS Command
010	0210	AVAILABLE Command
021	0221	COMPARE CONTROLLER DATA Command
040	0240	COMPARE HOST DATA Command
013	0213	DETERMINE ACCESS PATHS Command
022	0222	ERASE Command
023	0223	FLUSH Command
002	0202	GET COMMAND STATUS Command
003	0203	GET UNIT STATUS Command
011	0211	ONLINE Command
041	0241	READ Command
024	0224	REPLACE Command
004	0204	SET CONTROLLER CHARACTERISTICS Command
012	0212	SET UNIT CHARACTERISTICS Command
042	0242	WRITE Command

FLAG	DESCRIPTION
0200	Bad Block Reported
0100	Bad Block Unreported
0040	Error Log Generated
0020	Serious Exception

CODE	STATUS/EVENT CODE DESCRIPTION
(000)	"Success"
0	Normal
040	Spin-down Ignored
0100	Still Connected
0200	Duplicate Unit Number
0400	Already Online
01000	Still Online
(001)	"Invalid Command"
01	Invalid Message Length
*	Any other code points to command message field in error
(002)	"Command Aborted"
	Sub-codes are not used.

CODE	STATUS/EVENT CODE DESCRIPTION
(003)	"Unit-Offline"
03	Unit unknown or online to another controller
043	No volume mounted or drive disabled via RUN/STOP switch
0103	Unit is inoperative
0203	Duplicate unit number
0403	Unit disabled by field service or internal diagnostic
(004)	"Unit-Available"
	Sub-codes are not used.
(005)	"Media Format Error"
0005	"Data Error" accessing RCT or FCT Sector was written with "Force Error" modifier
0105	"Data Error" accessing RCT or FCT Invalid header
0145	"Data Error" accessing RCT or FCT Data Sync not found (Data Sync timeout)
0245	Disk is not formatted with 512 byte sectors
0305	Disk is not formatted or FCT corrupted
0345	"Data Error" accessing RCT or FCT Uncorrectable ECC Error
0405	RCT corrupted
(006)	"Write Protected"
020006	Unit is Hardware Write Protected
010006	Unit is Software Write Protected
(007)	"Compare Error"
	Sub-codes are not used.

F-4 MSCP Error Codes

CODE	STATUS/EVENT CODE DESCRIPTION
(010)	"Data Error"
0010	Sector was written with "Force Error" modifier
0110	Invalid header
0150	Data Sync not found (Data Sync timeout)
0210	Correctable error in ECC field
0350	Uncorrectable ECC Error
0410	One Symbol ECC Error
0450	Two Symbol ECC Error
0510	Three Symbol ECC Error
0550	Four Symbol ECC Error
0610	Five Symbol ECC Error
0650	Six Symbol ECC Error
0710	Seven Symbol ECC Error
0750	Eight Symbol ECC Error
(011)	"Host Buffer Access Error"
011	Host buffer access error, cause not available
051	Odd transfer address
0111	Odd byte count
0151	Non-existent memory error
0211	Host memory parity error
(012)	"Controller Error"
052	SERDES overrun or underrun error
0112	EDC Error
0152	Inconsistent internal data structure
0212	Internal EDC error
0412	Controller overrun or underrun
0452	Controller memory error
(013)	"Drive Error"
053	Drive command time out
0113	Controller detected transmission error
0153	Positioner error (mis-seek)
0213	Lost read/write ready during or between transfers
0253	Drive clock dropout
0313	Lost receiver ready for transfer
0353	Drive detected error
0413	Controller detected pulse or state parity error
0513	Controller detected protocol error
0553	Drive failed initialization
0613	Drive ignored initialization
0653	Receiver Ready collision

Appendix G

Rabads Program Example

This sample rabads session illustrates:

- Booting the rabads program
- Obtaining on-line help information
- Replacing a dynamic bad block
- Exiting the rabads program

A sample response follows each prompt. If no response is given, the default is assumed.

G-2 Rabads Example

This example is of booting the stand-alone rabads program and obtaining on-line rabads help information:

```
Boot: ra(0,0)/sas/rabads

ULTRIX-11 MSCP Disk Initialization Program

rabads <help exit drives status table init replace>: h

To correct typing mistakes, press <DELETE> to erase one character
or <CTRL/U> to erase the entire line.

To execute a command, type the first letter of the command then
press <RETURN>.  The program may prompt for additional information.

The valid RABADS commands are:

help    - Print this help message.

exit    - Exit from the RABADS program.

drives  - List the disks that can be initialized with RABADS.

status  - Print the status and geometry of the specified disk.

table   - Print the RCT (Revector Control Table) for a disk.

init    - Do a read scan of the disk and replace any bad blocks.

replace - Force replacement of a disk block.
```

This example is of replacing a reported bad block on a RA81 disk and exiting the rabads program:

```
rabads <help exit drives status table init replace>: r

Disk type < ra60 ra80 ra81 rx50 rd51 rd52 rc25 >: ra81

Unit number < 0-7 >: 1

Block number: 40000

Block 40000 read check: SUCCEEDED - bad block reported

Block: 40000 - replacement 4 5 6 7 8 9 10 11 12 13 SUCCEEDED!

rabads <help exit drives status table init replace>: e
Exit called

Boot:
```

Appendix H

ULTRIX-11 User Device Driver Commentary

This appendix contains a printout of the ULTRIX-11 User Device Driver Code. This is intended both as the main source of documentation as well as the prototype code that you are to modify when writing your own device driver.

```
/************************************************************************
 *                                                                      *
 *                    Copyright (c) 1984 by                             *
 *            Digital Equipment Corporation, Maynard, MA                *
 *                    All rights reserved.                              *
 *                                                                      *
 *   This software is furnished under a license and may be used and     *
 *   copied only in accordance with the terms of such license and       *
 *   with the inclusion of the above copyright notice.  This            *
 *   software or any other copies thereof may not be provided or        *
 *   otherwise made available to any other person.  No title to and      *
 *   ownership of the software is hereby transferred.                   *
 *                                                                      *
 *   This software is derived from software received from the           *
 *   University of California, Berkeley, and from Bell                  *
 *   Laboratories.  Use, duplication, or disclosure is subject to       *
 *   restrictions under license agreements with University of           *
 *   California and with AT&T.                                          *
 *                                                                      *
 *   The information in this software is subject to change without      *
 *   notice and should not be construed as a commitment by Digital      *
 *   Equipment Corporation.                                             *
 *                                                                      *
 *   Digital assumes no responsibility for the use or reliability       *
 *   of its software on equipment which is not supplied by Digital.     *
 *                                                                      *
 ************************************************************************/

/*
 * ULTRIX-11 V2.0 - Prototype User Device Driver Commentary
 *
 *  0.0   ABOUT THIS DOCUMENT
 *
 * This document is intended to act as the documentation for the prototype
 * ULTRIX-11 user-written device driver. The various sections of the text
 * below serve to familiarize the writer of a device driver with the
 * environment, interface, and requirements of a device driver.  This is
 * in no way complete -- no such claim is made.
 *
 * The format of the documentation is (hopefully) such that the time
 * required to write a driver should not be great, with the majority of
 * the necessary information available within this file.
 *
 * The following sections from the ULTRIX-11 Documentation are recommended
 * reading and contain information concerning the ULTRIX-11 I/O system.
 *
 *      System Management Guide, Volume 1
 *              Chapter 1: ULTRIX-11 I/O SYSTEM
 *
 *      Programmer's Manual, Volume 2A
 *              Section on "The UNIX Time-Sharing System"
 *              Section on "UNIX implementation"
 *              Section on "UNIX I/O System"
 *
```

* The following information is available within this document:
*

```
 *          3.9.1       Allocating Unibus Map Registers
 *          3.9.2       Deallocating Unibus Map Registers
 *         3.10        Saving and restoring floating point context
 *          3.10.1      Saving floating point context
 *          3.10.2      Restoring floating point context
 *         3.11        Logging device errors
 *         3.12        Printing messages on the console
 *
 *        4.0         RANDOM CAUTIONS
 *         4.1         Use of the floating point processor
 *         4.2         Never attempt to suspend processing at interrupt
 *                         level
 *         4.3         Provide for sleep()/wakeup() symmetry
 *         4.4         Use of elevated priorities
 *         4.5         Use of the Unibus Map
 *
 *        5.0         DATA STRUCTURES
 *         5.1         Device CSR assignments
 *         5.2         Unibus address extension register
 *         5.3         Device CSR structure definitions
 *         5.4         Structures for drivers requiring terminal structures
 *         5.5         Local buffers requiring static Unibus Map assignment
 *         5.6         Cells identifying available hardware
 *
 *        ...         PROTOTYPE DEVICE DRIVER CODE SEGMENTS
 *
 * This prototype file can be used to produce a device driver; note that
 * definitions beginning with XX should be changed to the appropriate
 * two character prefix for the specific device.  The ULTRIX-11 system
 * allows for the inclusion of up to four user-written device drivers
 * with the system, named u1, u2, u3, and u4.  Within this file, all
 * XX references should be changed to u1, u2, u3, or u4.  This is YOUR
 * choice, but under no circumstances could there be two drivers for
 * the same uX device.
 *
 */

/*
 *  1.0   OVERVIEW OF THE DRIVER ENVIRONMENT
 *
 * There are two categories of drivers in the ULTRIX-11 system,
 * character oriented (i.e. terminals and printers), and block
 * oriented (i.e. disks and tapes).  The kernel interface to these
 * drivers differs inte device characteristics and capabilities.
 * The following documentation serves to familiarize the first
 * time driver coder with a summary of the interface, and the
 * provided support within the kernel.
 *
 *  1.1   USER INFORMATION PASSED TO THE DEVICE DRIVER
 *
 * When a user's request is passed to a device driver, certain
 * fields within the user structure have been initialized by the
 * kernel for use by the driver to process the request.
 *
```

```
* Basically, the following information is available:
*
*      u.u_base   - base virtual address of the buffer
*      u.u_count  - size of the buffer
*      u.u_offset - byte file address of the data to read
*      u.u_segflg - indicates whether u_base is in user
*                   or kernel mode (0=user)
*      u.u_arg[]  - three words of user arguments (passed
*                   by ioctl())
*
* NOTE: Kernel support for types of devices varies.  For
*       example, terminal device drivers are under the
*       control of the higher level terminal driver.  This
*       makes it unnecessary for the terminal device driver
*       to use the user structure cells.
*       Disk devices also behave in a special manner.  Most
*       I/O to/from disks occur using system buffers.  As a
*       result, it is also unnecessary for disk drivers to
*       use the user structure cells.
*       The descriptions of the cells are included here for
*       those drivers which do not fall into either the
*       terminal or disk/magnetic tape categories.
*
*  1.2   NOTES ABOUT DRIVER CODE RUNNING AT INTERRUPT LEVEL
*
* INTERRUPT LEVEL is defined as the state whereby a device has
* interrupted the operations of a user or kernel-mode process
* with a request for service by the driver code.  Interrupts can
* occur at any time, so long as the current processor priority is
* less than that of the interrupting hardware device.  When in this
* state, it is possible (and quite probable) that the user who
* may have issued the currently active request against the device
* is not the currently mapped user (interrupts normally occur as
* part of device processing, typically signaling a change in
* device status, or the completion of an operation).
* As a result, the parameters contained in the user structure may
* not those of the requestor.  It is therefore imperative that the
* device driver do not access the currently mapped user structure.
*
*              * * *   WARNING   * * *
*      Do not, UNDER ANY CIRCUMSTANCES, issue a
*      sleep() while at INTERRUPT LEVEL! To do so
*      would certainly invite a system crash!
*
*
*  2.0   DEVICE DRIVER OPERATIONS
*
* The functions supported by device drivers differ between character
* and block oriented devices.  The following sections describe the
* various entry points, parameters, and system routines available to
* drivers.
*
*  2.1   CHARACTER DEVICE PROCESSING SPECIFICS
*
```

```
 * Devices which are character oriented are interfaced to from the
 * kernel via a data structure "cdevsw".  Also, block devices will
 * be called at some of these entry points if access to the device
 * is in "raw" mode (physical access to the device). The "cdevsw"
 * structure defines the addresses of the various driver action
 * routines:
 *
 * .d_open - address of driver's open(2) routine
 *      open(dev,flag)
 *              dev = device major/minor number
 *              flag = 0 for read, non-zero for write (-1 indicates
 *                      that the kernel is calling as part of system
 *                      initialization/startup)
 *
 * .d_close - address of driver's close(*) routine
 *      close(dev,flag)
 *              dev = device major/minor number
 *              flag = 0 indicates device open read-only; non-zero
 *                      indicates device was open read/write
 *              NOTE: Only the last process to close(2) will result
 *                    in driver notification.
 *
 * .d_read - address of driver's read(2) routine
 *      read(dev)
 *              dev = device major/minor number
 *              also uses fields described above:
 *                      u.u_segflag
 *                      u.u_base
 *                      u.u_count (for read, this will never be zero)
 *                      u.u_offset
 *
 * .d_write - address of driver's write(2) routine
 *      write(dev)
 *              dev = device major/minor number
 *              also uses fields described above:
 *                      u.u_segflg
 *                      u.u_base
 *                      u.u_count (for write, this may be zero)
 *                      u.u_offset
 *
 * .d_ioctl - address of driver's ioctl(2) routine
 *      ioctl(dev,v)
 *              dev = device major/minor number
 *              v = vector for arguments;
 *                      for gtty(), v is vector to return 3 words,
 *                      for stty(), v = 0, and u_arg[0..2] contain
 *                              up to 3 argument words
 *
 * .d_stop - address of driver's "stop" routine
 *      stop(dev)
 *             dev = device major/minor number
 *
 *      NOTE: The stop() routine is specifically for terminal
 *            devices.  It is used to perform the ^S function,
```

```
*            which stops the output going to the terminal.
*
* .d_ttys - address of first tty structure for this driver
*      struct tty **d_ttys
*
*
*  2.2   BLOCK DEVICE PROCESSING SPECIFICS
*
* Block oriented devices require more support within the system.
* Since block devices contain file systems, several useful routines
* are available to the block device driver to aid in the maintenance
* of the file system, and to enhance performance.  Terminal devices
* are controlled by a single terminal driver which in turn requests
* the actual I/O operations from the various "controller" drivers,
* such as the DH11 or DZ11 multiplexer.  In much the same way, the
* file system support routines, swapper, and others, may issue a
* request against a block driver which is written for a specific
* controller.  This request is received by a driver using a structure
* known as a "buffer header".  This structure contains all information
* necessary for the driver to perform the actual disk (or tape)
* operation.  The buffer header is described later.  Since the
* buffer contains the operation code, only one entry point is required
* for a block device driver: the strategy routine.  The driver may
* include separate read() and write() routines.  However, these
* routines will only be called by the kernel to perform physical device
* access independent of the file structure. In general, file structured
* device access will be accomplished by the kernel calling the device
* strategy routine.  Drivers providing physical device access should
* create buffer headers whose target buffer address is within the
* caller's address space.  The buffer headers can then be passed to
* physio(), which in turn will call the strategy routine.
*
*  2.2.1 BLOCK DRIVER INTERFACE
*
* Devices which are block oriented are interfaced to from the
* kernel via a data structure "bdevsw".  This structure defines
* the addresses of the various driver action routines:
*
* .d_open - address of driver's open(2) routine
*      (see .d_open for CHARACTER devices)
*
* .d_close - address of driver's close(*) routine
*      (see .d_close for CHARACTER devices)
*
* .d_strategy - address of driver's strategy (I/O) routine
*      This routine is unlike the CHARACTER device I/O routines.
*      It is called by the kernel, with the address of a kernel
*      buffer header.  The header contains sufficient information
*      for the disk driver to be able to complete the request.
*      The format of the buffer header is described below.
*
*  2.2.2 KERNEL BUFFERS
*
* Kernel buffers are portions of kernel-controlled memory set aside
```

```
 * for disk data buffering.  They are used by block-oriented drivers
 * in order to provide the user with byte-oriented I/O similar to
 * that provided by character-oriented devices.  This is limited to
 * only that data which is contained within a file (direct device
 * access is device specific, and generally block-oriented devices
 * must perform I/O in block units, with transfers occurring on word
 * boundaries).  The user (requestor) can either be a user process,
 * or even be the kernel itself.  For example, the kernel
 * requires disk I/O in order to maintain the disk file structures,
 * and to provide support for process swapping (note that the kernel
 * does not use buffered I/O for process swapping; this note serves
 * to indicate that the kernel does perform I/O internally).
 *
 * Each kernel buffer has an associated buffer header.  The header
 * contains a full set of context concerning the buffer, and the
 * device which the buffer contains a copy of data from.  The vast
 * majority of requests for disk data will come in the form of
 * kernel buffers.  Since the buffer itself specifies the actual
 * operation code (read vs. write), there is no need to supply two
 * different disk I/O action routines.  Also, since much of the
 * processing is the same in either case, there is one single entry
 * point into a disk driver to perform I/O.  This routine is called
 * a strategy routine.  It may call separate read() or write()
 * routines, but in general, the kernel calls the single strategy
 * routine.
 *
 * The following is a list of fields from the buffer header which
 * may be useful to the BLOCK device driver:
 *
 * .b_flags - flags describing the status (and I/O function code)
 *      B_WRITE   - buffer should be written to disk (= ~B_READ)
 *      B_READ    - disk block should be read into buffer
 *      B_DONE    - operation complete, buffer is intact
 *      B_ERROR   - error on operation, buffer may be bogus
 *      B_BUSY    - buffer is inuse, and not available
 *      B_PHYS    - physical I/O
 *      B_MAP     - Unibus Map allocated for this buffer
 *      B_WANTED  - someone waiting for this buffer, use wakeup()
 *      B_AGE     - upon release, place at end of free list
 *      B_ASYNC   - don't stall caller for I/O completion
 *      B_DELWRI  - write block only when necessary (retirement)
 *      B_TAPE    - magnetic tape block, don't delay writes
 *      B_MOUNT   - buffer is superblock from a mounted device
 *
 * .b_dev - major/minor number of the device which this buffer is from
 *
 * .b_bcount - count of bytes to transfer
 *
 * .b_addr - low order core address
 *
 * .b_blkno - block number within the partition
 *
 * .b_xmem - high order core address
 *
```

```
* .b_error - I/O status code returned on I/O
*
* .b_resid - words not transferred on error
*
* The following fields are used when performing DMA I/O to/from
* a device when on a 22-bit system with a Unibus Map:
*
* .b_raddr - low order core address
*
* .b_rxmem - high order core address
*
* (When the Unibus Map is allocated, the I/O operation is bound
*  to an 18-bit address range which may or may not be the same
*  as the "real" memory address. This address is used by the
*  Unibus Map for translation into the "real" 22-bit address.
*  Therefore, the original values of .b_addr and .b_xmem have
*  been saved in .b_raddr and .b_rxmem accordingly.)
*
*  2.2.3 USE OF THE UNIBUS MAP IN 22-BIT SYSTEMS (WITH UBM)
*
* On 22-bit systems with a Unibus Map, the kernel requires the
* use of specific UMR (Unibus Map Register) allocation and
* deallocation routines in order to allow multiple UBM users
* to "share" the UMRs.  The 32 UMRs are assigned as follows:
*
*       UMRs   Usage by kernel and drivers
*      ------  ------------------------------------------------
*           0  Statically assigned to the bottom of BSS
*              space, where device control buffers (MSCP, such
*              as the TS11, DH11, and the RA class of disks)
*              are allocated (see section 5.5 for details
*              on how to allocate space within this UMR).
*         1..9 Permanently mapped to I/O buffers used by the
*              kernel (see section 2.2.2 for details on kernel
*              buffers)
*       10..30 Dynamically allocated in groups of 7 UMRs in
*              order to perform raw I/O operations directly
*              to the location specified by the requestor.
*              The three groups span UMRs 10..16, 17..24,
*              and 25..30.
*           31 Not available (as defined by the architecture
*              of the PDP-11).
*
* Special routines exist for sharing the Unibus Map.  They are
* described in section 3.9.
*
*
*  2.3   INTERFACING A COMMUNICATIONS DEVICE TO THE
*          TERMINAL DRIVER
*
* The environment of the terminal oriented device varies slightly
* from those which are obviously not terminals.  Normally, a
* device driver's action routine is dealt with directly by the
* kernel.  In the case of terminals, there is more that must be
```

```
 * done.  Each terminal has associated with it a set of characteristics
 * and a "line discipline".  The characteristics are accessible via
 * the stty(1) command, or the ioctl(2) or gtty(2)/stty(2) system
 * calls.  They are used to control the application of certain types
 * of processing which has been selected (or defaulted) by the user.
 * The line discipline is a set of common processing that terminal
 * devices share.  It is a set of extensions to the device specific
 * action routines.  The line discipline is implemented in software
 * called the "terminal driver".  It consists of a set of I/O routines
 * which are device independent, and which interface in some common
 * way (via the cdevsw and linesw structures/tables) to the actual
 * device driver routines.
 *
 * When a process requests that a character be transmitted to the
 * terminal, the terminal driver preprocesses that character based
 * upon the characteristics defined in the tty structure for the
 * device.  The resulting character(s) are placed in an output queue
 * and the device driver is called at the XXstart() routine to
 * initiate that actual transmission.  In this situation, the terminal
 * driver is acting as master, with the device driver acting as a
 * slave.  The terminal driver uses the cdevsw structure to call
 * the device driver at the XXstart() entry point.
 *
 * When a device driver receives a character from the device, it must
 * likewise pass off the character to the terminal driver to be
 * preprocessed according to the tty characteristics, and finally
 * placed in a queue for the process accessing the terminal.  The
 * process can obtain the characters in the queue by a read(2) system
 * call.  This call enters the device driver at the XXread() entry
 * point.  In order for the terminal driver to return the characters,
 * the device driver must call into the terminal driver's read()
 * routine.  This is done through the linesw structure.  In this
 * situation, the device driver acts as the master and the terminal
 * driver acts as the slave.
 *
 * There is an entry in the linesw structure for each possible
 * interaction between the device and terminal driver.
 * Also, there is an entry in the cdevsw structure for each possible
 * interaction between the terminal and device driver.
 *
 * A description of the calling convention through the linesw structure
 * is show below.  In each case, the device driver has been called
 * at one of the action routines, and it in turn is passing control to
 * the equivalent device independent terminal driver routine.
 *
 * File /usr/include/sys/conf.h contains the description of the linesw
 * structure, and performs an "extern struct linesw" inline.
 *
 * In all of the following, argument 'tp' is the address of the tty
 * structure which is to be acted upon.  Other arguments are routine
 * specific.
 *
 * When the device driver is called at XXopen(), it should call the
 * terminal driver as follows:
```

```
*
*     (*linesw[tp->t_line].l_open)(tp)
*
*       struct tty *tp;
*
* When the device driver is called at XXclose(), it should call the
* terminal driver as follows:
*
*     (*linesw[tp->t_line].l_close)(tp)
*
*       struct tty *tp;
*
* When the device driver is called at XXioctl(), it should call the
* terminal driver as follows:
*
*     (*linesw[tp->t_line].l_ioctl)(tp, com, addr, flag)
*
*       struct tty *tp;
*       int com;
*       caddr_t addr;
*       int flag;
*
* When the device driver is called at XXread(), it should call the
* terminal driver as follows:
*
*     (*linesw[tp->t_line].l_read)(tp)
*
*       struct tty *tp;
*
* When the device driver is called at XXwrite(), it should call the
* terminal driver as follows:
*
*     (*linesw[tp->t_line].l_write)(tp)
*
*       struct tty *tp;
*
* When the device driver is called via an interrupt at XXrint(), it
* should call the terminal driver as follows, to pass the character:
*
*     (*linesw[tp->t_line].l_rint)(c, tp)
*
*       char c;
*       struct tty *tp;
*
*
*  3.0   KERNEL ROUTINES AVAILABLE TO DRIVERS
*
* The ULTRIX-11 kernel contains some useful routines which can
* by used by the device driver during normal operations.
* Some routines are specific to either a character or block
* oriented device, and are typically used by the device drivers
* provided by DIGITAL.
* A description of the actual routine, its parameters, and the
* guidelines governing their use is provided.
```

```
*
* The type definitions can be found in /usr/include/sys/types.h.
*
*  3.1   MOVING DATA TO/FROM A USER'S BUFFER
*
*        Routine iomove() is used to transfer byte aligned
*        data into or out of the requestor's buffer area.
*        The requestor's buffer is described by the various
*        fields in the currently mapped user context area.
*
*          iomove(cp, n, flag)
*
*            caddr_t cp;
*            int n;
*            int flag;
*
*            cp:       address of buffer in requestor's mode to move
*            n:        number of bytes to transfer
*            flag:     B_READ to read (copy) from the requestor's buffer
*                      B_WRITE to write (copy) to the requestor's buffer
*
*        The following fields are used as input:
*
*          u_segflg:  indicates whether the requestor is kernel or user
*          u_base:    base address of requestor's buffer in the mode
*                     specified by u_segflg
*          u_count:   number of bytes to transfer
*
*        The following fields are returned as output:
*
*          u_error:   not initialized, but may return EFAULT if the
*                     requestor's buffer is not mappable
*          u_base:    updated to point past the data moved
*          u_offset:  updated to point past the data moved
*          u_count:   updated to reflect the bytes moved
*
*  3.2   SCHEDULING TIME BASED CALLS FROM THE KERNEL
*
*        Routine timeout() is used to schedule periodic calls from
*        the kernel.  Note that the issuer of timeout() is called
*        at the clock's priority (PR6) by the code processing the
*        clock's interrupts.  Time is specified in "ticks", i.e.
*        clock interruptions (this is the line frequency, typically
*        either 50 or 60 HZ, therefore, 50 or 60 ticks per second).
*
*          timeout(fun, arg, tim)
*
*            int (*fun)();
*            caddr_t arg;
*            int tim;
*
*            fun:      address of the routine to be called
*            arg:      the argument to be passed to the routine
*            tim:      the number of ticks to wait for prior to
```

```
*                    calling the specified routine (interval
*                    MUST be in the range of 0 to 32767)
*
*       The user structure is NOT used for input or output.
*
*  3.3  PROCESS CONTROL
*
*       Routines exist to perform two basic scheduling operations:
*       sleep() to suspend a process, and wakeup() to resume a
*       suspended process.
*
*  3.3.1 SUSPENDING THE EXECUTION OF THE CURRENT PROCESS
*
*       Routine sleep() is used to place the currently mapped
*       user into sleep (i.e. not runnable) state.  This causes
*       the scheduler to select another process for execution.
*       THIS ROUTINE SHOULD NOT BE CALLED FROM INTERRUPT LEVEL.
*       Processes which are sleeping on an event have a status
*       of "SWAIT".  Also, it is important that the caller check
*       to see if conditions are favorable for continuation after
*       the sleep() has returned.
*
*         sleep(chan, pri)
*
*           caddr_t chan;
*           int pri;
*
*           chan:    value which serves to identify the reason
*                    as to why a process is waiting - there must
*                    be a corresponding routine which is expected
*                    to issue a wakeup(event) in order to bring
*                    the process out of sleep state, and make it
*                    runnable once again
*           pri:     is the priority to be given to the process
*                    when the eventual wakeup() is issued (this
*                    priority is the scheduling priority)
*                    "pri" is significant in that values greater
*                    than PZERO will allow signals to be delivered
*                    to the process
*
*         The user structure is NOT used for input or output.
*
*         NOTE: Calling sleep() with a wchan of zero will cause
*               the system to "panic".
*
*         NOTE: A general 4-second timeout routine is available.
*               The caller need only call sleep(&lbolt,pri).
*               (&lbolt is the identifier of an event which
*               occurs every 4 seconds.)
*
*  3.3.2 ACTIVATING A PROCESS WHICH HAS BEEN SUSPENDED
*
*       Routine wakeup() is used to activate any process waiting
*       for a specific resource.  Generally, the argument to the
```

```
*       wakeup() call is the identifier which connects the waiting
*       process to the event that it is waiting for.  The actual
*       routine will search the process list for all processes which
*       are waiting for the specified event.  If any matches occur,
*       the process status is set to "SRUN", and the wait mask is
*       cleared.  The process is then eligible for scheduling based
*       upon the priority specified in the sleep() request which
*       placed it into the suspended state in the first place.
*
*         wakeup(chan)
*
*           caddr_t chan;
*
*           chan:    identifier of the event which has come to
*                    pass (this event identifier should match
*                    that specified by a previous sleep()
*
*       The user structure is NOT used for input or output.
*
*  3.4  DYNAMICALLY ALTERING PROCESSOR PRIORITY
*
*       Several routines are available to change the current
*       priority of the processor.  All routines except splx()
*       do not accept arguments, and return the processor
*       status word (PS, which includes a field indicating the
*       processor's priority).  Routine splx() sets the current
*       processor priority, but does not return a PS.
*       The normal procedure to use an elevated priority is to
*       use one of the explicit priority routines, saving the
*       PS which they return, and to use the saved PS later
*       as the argument to splx() to return the processor to
*       the original priority.
*
*         spl0()   {set processor to priority 0}
*         spl1()   {set processor to priority 1}
*         spl4()   {set processor to priority 4}
*         spl5()   {set processor to priority 5}
*         spl6()   {set processor to priority 6}
*         spl7()   {set processor to priority 7}
*
*         splx(ps)
*
*           int ps;
*
*           ps:    processor status word returned by one of the
*                  explicit splN() routines
*
*        The user structure is NOT used for input or output.
*
*        NOTE: Care must be taken when using changing the priority
*              of the processor!  The purpose of these routines is
*              to prevent the execution of event-driven kernel code
*              so as to guarantee serialized access to sensitive
*              kernel data in a controlled manner.  Execution of
```

```
*               large amounts of code at an elevated priority will
*               decrease the performance of the system overall, and
*               if not used properly, can place the processor into
*               an undetermined state.
*
*  3.5   MOVING DATA BETWEEN PROCESSOR MODES (WORD, BYTE)
*
*        Several routines are available which can be used to read or
*        write data between the current processor mode, and the
*        previous processor mode.  The current processor mode is
*        almost always KERNEL, and the previous is typically USER.
*        There are eight routines available.  Four are used to read
*        from previous mode, and four to write.  Two of the four
*        are used to select byte or word alignment, and the remaining
*        two are used to specify whether the previous mode's
*        instruction or data space is to be used.
*
*        The user structure is NOT used for input or output.
*
*        NOTE: All of these routines return a value of -1 if the
*              address specified is nonexistent.  Therefore, care
*              should be taken when using these routines to
*              differentiate between a successfully read data item
*              of -1, and a non-existent memory trap return of -1.
*
*  3.5.1 FETCH PREVIOUS DATA SPACE WORD
*
*        Routine fuword() is used to read a word from the previous
*        mode's data space.
*
*          fuword(address)
*
*          int *address;
*
*          address: address of the word to be fetched from the
*                   the previous mode's data space
*
*        Since this routine uses the MFPD instruction, and is quite
*        slow, it is NOT recommended that large amounts of data be
*        moved using this routine.
*
*  3.5.2 FETCH PREVIOUS DATA SPACE BYTE
*
*        Routine fubyte() is used to read a byte from the previous
*        mode's data space.
*
*          fubyte(address)
*
*          char *address;
*
*          address: address of the byte to be fetched from the
*                   the previous mode's data space
*
*        Since this routine uses the MFPD instruction, and is quite
```

```
*       slow, it is NOT recommended that large amounts of data be
*       moved using this routine.
*
*  3.5.3 FETCH PREVIOUS INSTRUCTION SPACE WORD
*
*       Routine fuiword() is used to read a word from the previous
*       mode's instruction space.
*
*         fuiword(address)
*
*         int *address;
*
*         address: address of the word to be fetched from the
*                  the previous mode's instruction space
*
*       Since this routine uses the MFPI instruction, and is quite
*       slow, it is NOT recommended that large amounts of data be
*       moved using this routine.
*
*  3.5.4 FETCH PREVIOUS INSTRUCTION SPACE BYTE
*
*       Routine fuibyte() is used to read a byte from the previous
*       mode's instruction space.
*
*         fuibyte(address)
*
*         char *address;
*
*         address: address of the byte to be fetched from the
*                  the previous mode's instruction space
*
*       Since this routine uses the MFPI instruction, and is quite
*       slow, it is NOT recommended that large amounts of data be
*       moved using this routine.
*
*  3.5.5 STORE PREVIOUS DATA SPACE WORD
*
*       Routine suword() is used to write a word into the previous
*       mode's data space.
*
*         suword(address, data)
*
*         int *address;
*         int data;
*
*         address: address where the word 'data' is to be stored in
*                  the previous mode's data space
*         data:    word to be stored at 'address' in the previous
*                  mode's data space
*
*       Since this routine uses the MTPD instruction, and is quite
*       slow, it is NOT recommended that large amounts of data be
*       moved using this routine.
*
```

```
*  3.5.6 STORE PREVIOUS DATA SPACE BYTE
*
*       Routine subyte() is used to write a byte into the previous
*       mode's data space.
*
*         subyte(address, data)
*
*         char *address;
*         char data;
*
*         address: address where the byte 'data' is to be stored in
*                  the previous mode's data space
*         data:    byte to be stored at 'address' in the previous
*                  mode's data space
*
*       Since this routine uses the MTPD instruction, and is quite
*       slow, it is NOT recommended that large amounts of data be
*       moved using this routine.
*
*  3.5.7 STORE PREVIOUS INSTRUCTION SPACE WORD
*
*       Routine suiword() is used to write a word into the previous
*       mode's instruction space.
*
*         suiword(address, data)
*
*         int *address;
*         int data;
*
*         address: address where the word 'data' is to be stored in
*                  the previous mode's instruction space
*         data:    word to be stored at 'address' in the previous
*                  mode's instruction space
*
*       Since this routine uses the MTPI instruction, and is quite
*       slow, it is NOT recommended that large amounts of data be
*       moved using this routine.
*
*  3.5.8 STORE PREVIOUS INSTRUCTION SPACE BYTE
*
*       Routine suibyte() is used to write a byte into the previous
*       mode's instruction space.
*
*         suibyte(address, data)
*
*         char *address;
*         char data;
*
*         address: address where the byte 'data' is to be stored in
*                  the previous mode's instruction space
*         data:    byte to be stored at 'address' in the previous
*                  mode's instruction space
*
*       Since this routine uses the MTPI instruction, and is quite
```

```
*       slow, it is NOT recommended that large amounts of data be
*       moved using this routine.
*
*  3.6  MANIPULATION OF "CLISTS" FOR CHARACTER DEVICES
*
*       Character oriented devices store spans of characters in
*       structures called CLISTS.  Two special routines are
*       used to retrieve and to store a character from and into
*       a CLIST.  Both require an argument which specifies the
*       address of the device's CLIST list pointers.
*
*       CLISTS are described in /usr/include/sys/clist.h.
*
*  3.6.1 PLACING A CHARACTER INTO A CLIST
*
*       Routine putc() is used to append a character onto a
*       clist.  If there is insufficient space in the clist for
*       the character, the routine returns a -1.
*
*         putc(c, p)
*
*           int c;
*           struct clist *p;
*
*           c:      character to be placed into the clist
*           p:      address of the clist list pointers for
*                   the device whose clist is to be used
*
*       The user structure is NOT used for input or output.
*
*  3.6.2 REMOVING A CHARACTER FROM A CLIST
*
*       Routine getc() is used to obtain the first character
*       from a clist.  If there are no more characters in the
*       clist, the routine returns a -1.
*
*         getc(p)
*
*           struct clist *p;
*
*           p:      address of the clist list pointers for
*                   the device whose clist is to be used
*
*       The user structure is NOT used for input or output.
*
*  3.7  BUFFERED I/O SUPPORT ROUTINES
*
*       Several routines within the kernel are available for
*       buffer management.  While their use by the typical
*       driver will be minimal (save for brelse()), their
*       description is aimed at increasing the understanding
*       of the ULTRIX-11 block I/O subsystem.
*       Note that the routines requesting a read of data will
*       stall (i.e. not return) until the request has been
```

```
*       completed.  It is important that the caller check the
*       status (which is eventually) returned by the driver
*       after issuing the request.
*       The various read routines (bread(), breada(), and
*       getblk()) return pointers to the buffer headers which
*       describe the requested data.
*       When buffers are returned by bread(), breada(), and
*       getblk(), the buffer headers are "locked down", in
*       order to prevent the simultaneous access by others
*       (B_BUSY is turned on).
*
*  3.7.1 BUFFERED READ, SYNCHRONOUS
*
*       Routine bread() is used to allocate, read, and return
*       a block of disk data.  The presence of the block (or
*       an error indication in the event of a failure) is
*       guaranteed.
*
*         bread(dev, blkno)
*
*           dev_t dev;
*           daddr_t block;
*
*           dev:    device major/minor number
*           blkno:  logical block number to be read
*
*       The user structure is NOT used for input or output.
*
*  3.7.2 BUFFERED READ, ASYNCHRONOUS
*
*       Routine breada() is used to allocate, read, and return
*       a block of disk data.  In addition, a read-ahead operation
*       will be performed on another disk block after the initial
*       request has been made available.
*       Note that only the first block will be "locked".  The second
*       block (rablkno) will be locked by one of the subsequent calls
*       by either bread(), getblk(), or another breada(), specifying
*       'rablkno' as the synchronous block to read.
*
*         breada(dev, blkno, rablkno)
*
*           dev_t dev;
*           daddr_t blkno;
*           daddr_t rablkno;
*
*           dev:    device major/minor number
*           blkno:  logical block number to be read
*           rablkno:logical block number to be read, after
*                   'blkno' has been read.
*
*       The user structure is NOT used for input or output.
*
*       Note that only the first block will be read and returned
*       by the breada() call.  A subsequent call using bread()
```

```
*       specifying 'rablkno' as the logical block can return the
*       address of the buffer header for the asynchronous read.
*       In that case, the caller would suspend execution pending
*       the completion of the asynchronous read operation if it
*       is still in progress (B_DONE is not set).
*
*  3.7.3 FIND, LOCK, AND RETURN BUFFERED BLOCK (IF PRESENT)
*
*       Routine getblk() is used to obtain the address of the
*       buffer header associated with a block of data providing
*       the block is resident within the system's buffers.  An
*       address of -1 indicates that the buffer is not present
*       in memory.  What differentiates this call from a bread()
*       call is that bread() guarantees that the data will be
*       read if it is not already resident, whereas getblk() may
*       return an error if the block is either not resident or
*       in the process of being read in.
*
*         getblk(dev, blkno)
*
*           dev_t dev;
*           daddr_t blkno;
*
*           dev:    device major/minor number
*           blkno:  logical block number to be read
*
*       The user structure is NOT used for input or output.
*
*  3.7.4 ALLOCATING AND INITIALIZING AN EMPTY BUFFER
*
*       Routine geteblk() is used to allocate a buffer and to
*       initialize it to NULLs.  The address of the buffer header
*       is returned by the call.
*
*         geteblk()
*
*       The user structure is NOT used for input or output.
*
*  3.7.5 RELEASING A LOCKED BUFFER
*
*       Routine brelse() is used to unlock a buffer which was
*       locked by a previous bread(), breada(), or getblk()
*       request.
*
*         brelse(dev, blkno)
*
*           dev_t dev;
*           daddr_t blkno;
*
*           dev:    device major/minor number
*           blkno:  logical block number to unlock
*
*       The user structure is NOT used for input or output.
*
```

In addition, routine brelse() performs the following:

- if B_WANTED is set (indicating that a process is sleeping elsewhere waiting for this buffer), then a wakeup() is issued
- the buffer is placed onto the free list- normally it is placed at the beginning, but if B_AGE is set, then it is placed onto the end (this will arrange for the buffer to NOT be reused immediately)
- bits B_WANTED, B_BUSY, B_ASYNC, and B_AGE are cleared

3.7.6 BUFFERED WRITE, SYNCHRONOUS

Routine bwrite() is used to write a block of buffered data to a device. The caller is suspended pending the completion of the write operation. When the operation completes, the status is returned to the user.

```
bwrite(bp)

  struct buf *bp;

  bp:     address of the buffer header of the block
          to be written
```

The user structure is NOT used as input.

The u_error cell is set to the I/O status code of the write operation.

3.7.7 BUFFERED WRITE, ASYNCHRONOUS

Routine bawrite() is used to write a block of buffered data to a device in an asynchronous manner. The caller is not suspended (return from bawrite() is immediate), and the user is not notified of the completion code. It is the responsibility of the caller to check the code and insure its propagation to the user if necessary.

```
bawrite(bp)

  struct buf *bp;

  bp:     address of the buffer header of the block
          to be written
```

The user structure is NOT used for input or output.

3.7.8 BUFFERED WRITE, DEFERRED

Routine bdwrite() is used to write a block of buffered data to a device in an asynchronous manner, but deferred until such time as it becomes necessary to write the block. The caller is not suspended (return from bdwrite()

```
*       is immediate), and the user is not notified of the
*       completion code.  It is the responsibility of the caller
*       to check the code and insure its propagation to the
*       user if necessary.
*
*         bdwrite(bp)
*
*           struct buf *bp;
*
*           bp:     address of the buffer header of the block
*                   to be written
*
*       The user structure is NOT used for input or output.
*
*  3.7.9 BUFFERED I/O COMPLETION
*
*       Routine iodone() is used to complete a buffered I/O
*       operation and to insure the necessary release of the
*       buffer or the awakening (via wakeup()) of the process
*       suspended for an I/O operation.  The routine can be
*       called whether the operation has completed with success
*       or not, and it is expected that the B_DONE and B_ERROR
*       flags have been maintained correctly.
*
*         iodone(bp)
*
*           struct buf *bp;
*
*           bp:     address of the buffer header whose I/O
*                   operation has completed
*
*       The user structure is NOT used for input or output.
*
*       Routine iodone() will perform several functions:
*
*             - if the buffer has an associated UBM allocation
*               (indicated by B_MAP), then mapfree() will be
*               called to unlock the allocated Unibus Map
*             - B_DONE is set, to indicate transaction complete
*             - if B_ASYNC is set, brelse() is called to release
*               the lock on the buffer
*             - [else] if B_WANTED is set, then a process is
*               sleeping waiting for the I/O to complete --
*               wakeup() is called
*
*               SEE THE DESCRIPTION OF BRELSE() IN SECTION 3.7.4.
*
*  3.7.10 PROPAGATION OF I/O STATUS TO THE REQUESTOR
*
*       Routine geterror() is used to return the I/O status
*       of an operation to the requestor.
*       The buffer header context is used to return the
*       status code to the user structure for completion
*       of an I/O operation.
```

```
*
*           geterror(bp)
*
*             struct buf *bp;
*
*             bp:     address of the buffer header whose status
*                     is to be propagated
*
*         The user structure is used to determine the mode of the
*         requestor, in order to return the status in the appropriate
*         place.
*
*   3.7A  PHYSICAL I/O
*
*         Block oriented devices can perform I/O in two manners.
*         Normally, I/O against a device is done in "file structured"
*         mode.  This means that the data being accessed on the disk
*         is coming from a disk partition, and specifically from an
*         opened file.  The file that has been opened is a block
*         special file from a specific disk and partition, and it
*         defines the areas of the disk which contain the file's data.
*         It is possible to access the disk (or partition) in "raw"
*         mode.  This would give the user access to the data within
*         the disk or partition independent of the file structure.
*         In this mode, it is required that the user read and write
*         data in disk block units, with the position always being
*         on a block boundary (512 byte boundary).
*
*         When performing physical I/O, the user is accessing the disk
*         via a character special file.  As a result, the cdevsw table
*         is being used instead of the bdevsw table to perform the
*         dispatching from the kernel to the driver.  Therefore, a
*         read(2) system call would cause the driver to be called at
*         the XXread() routine, and a write(2) system call would
*         cause the driver to be called at the XXwrite() routine.
*
*         A special routine is available to change the character based
*         operation into a buffered block operation.  The caller
*         (driver) should specify the address of a buffer header which
*         it owns (is part of the driver's space), and the address of
*         the strategy routine.  Physio() is used to accept these and
*         the device major/minor number and read/write (B_READ/B_WRITE)
*         flag, and to calculate and validate the caller's buffer
*         addresses.  It then calls the strategy routine to perform
*         a buffered read or write operation.
*
*           physio(strat, bp, dev, rw)
*
*               int (*strat)();
*               struct buf *bp;
*               int dev;
*               int rw;
*
*               strat:    address of the XXstrategy() routine
```

```
*             bp:       address of buffer header in the driver's space
*             dev:      device major/minor number
*             rw:       flag indicating the operation: B_READ to read
*                       from the disk, and B_WRITE to write to disk
*
*  3.8   READING/WRITING THE REQUESTOR'S BUFFER (BYTE ORIENTED)
*
*       Routine passc() is used to read or write the requestor's
*       buffer in a byte-by-byte fashion.  Only one routine
*       is used, the differentiation being whether an argument is
*       supplied or not.
*
*  3.8.1 READING A BYTE FROM THE REQUESTOR'S BUFFER
*
*       Routine passc() is used to read a byte from the requestor's
*       buffer.  The character is returned from the call, with a
*       value of -1 to indicate that the buffer is empty (i.e.
*       there is no more data to be read).
*
*         passc()
*
*       The following fields are used as input:
*
*         u_segflg:  indicates whether the requestor is kernel or user
*         u_base:    address of the next byte in requestor's buffer
*                    in the mode specified by u_segflg
*         u_count:   number of bytes remaining in requestor's buffer
*
*       The following fields are returned as output:
*
*         u_base:    updated to point to the next byte in buffer
*         u_count:   decremented to reflect number of bytes remaining
*
*  3.8.2 WRITING A BYTE (APPENDING) TO THE REQUESTOR'S BUFFER
*
*       Routine passc() is used to write (append) a byte to the
*       requestor's buffer.  The character is passed as the one
*       argument to the call, with a return value of -1 indicating
*       that the requestor's buffer is full (u_count has become zero
*       which indicates that no more data can be placed into the
*       buffer).
*
*         passc(c)
*
*           char c;
*
*       The following fields are used as input:
*
*         u_segflg:  indicates whether the requestor is kernel or user
*         u_base:    address of the next byte in requestor's buffer
*                    in the mode specified by u_segflg
*         u_count:   number of bytes available beginning at u_base to
*                    accept data
*
```

The following fields are used as output:

u_base: updated to point to next available (free) byte in the requestor's buffer
u_count: decremented to reflect the number of available bytes remaining in the requestor's buffer

3.9 SHARING THE UNIBUS MAP

For devices which accept buffer addresses in order to perform DMA (Direct Memory Access) I/O, there are two classes. The first are those devices which accept a full 22-bit address for the data transfers. These devices do not need to use the UBM routines. The second class are those devices which accept a 16-bit address, and two bits of "address extension" (referred to as bits A16 and A17 of the address). Under normal situations I/O to/from these devices would be limited to the lowest 18-bits of address space. With the UBM, it is possible to relocate 18-bit addresses into 22-bit addresses. There are a fixed number of UMRs available. In order to allow drivers to "share" these dynamically, several routines are available to the driver to allocate and deallocate them.

3.9.1 ALLOCATING UNIBUS MAP REGISTERS

Routine mapalloc() is used to allocate UMRs. If the necessary UMRs are not available, mapalloc() will sleep() and return when they are available.

```
mapalloc(bp)

struct buf *bp;
```

bp: address of the buffer header to allocate UMRs for

The user structure is NOT used as input or output.

NOTE: mapalloc() allocates and maps the UMRs to the requestor's buffers, and fields b_addr and b_xmem are used to describe the UMR allocation (b_addr is moved to b_raddr, and b_xmem is moved to b_rxmem)

3.9.2 DEALLOCATING UNIBUS MAP REGISTERS

Unibus Map Register deallocation is performed automatically by the routine iodone(), if B_MAP is set when it is called. Should the driver wish to deallocate the map itself, routine mapfree() can be used.

```
mapfree(bp)

struct buf *bp;
```

```
*
*          bp:    address of the buffer header whose UMR allocation
*                 should be released
*
*       The user structure is NOT used for input or output.
*
*  3.10  SAVING AND RESTORING FLOATING POINT CONTEXT
*
*       Section 4.1 contains a note concerning the use of the
*       floating point processor.
*
*       Two routines are available to save and restore the context
*       of the floating point unit for drivers which wish to use
*       the FPP.  The save area should be allocated from the driver's
*       address space and have a particular format.  The structure
*       definition of "fpsave" (below) shows the format of the save
*       area.
*
*            struct fpsave
*                   {
*                    int    XX_fps;          <-- FPP status register
*                    double XX_fpr[6];       <-- FPP registers
*                   };
*
*  3.10.1 SAVING FLOATING POINT CONTEXT
*
*       Routine savfp() can be used to save the current context of
*       the floating point processor.
*
*         savfp(savearea)
*
*         struct fpsave *savearea;
*
*       See the format of structure fpsave in section 3.10 above.
*
*       The user structure is NOT used for input or output.
*
*  3.10.2 RESTORING FLOATING POINT CONTEXT
*
*       Routine restfp() can be used to restore the previously saved
*       (savfp()) context of the floating point processor.  The
*       current context of the FPP is lost.
*
*         restfp(savearea)
*
*         struct fpsave *savearea;
*
*       See the format of structure fpsave in section 3.10 above.
*
*       The user structure is NOT used for input or output.
*
*  3.11  LOGGING DEVICE ERRORS
*
*       Currently there is no mechanism available to the user to
```

log device errors using the ULTRIX-11 error logging system. Instead, the user-written driver should call routine deverror() to display the error on the console.

```
deverror(bp, o1, o2)

struct buf *bp;
int o1;
int o2;
```

A call to deverror() will cause the following to appear on the console:

```
err <device>
blk=<blkno>, er=<o1>,<o2>
```

NOTE: deverror() uses printf(). See section 3.12 for information (and note) concerning the use of printf().

The user structure is NOT used for input or output.

3.12 PRINTING MESSAGES ON THE CONSOLE

Routine printf() can be used to print messages onto the ULTRIX-11 console terminal. This version of printf() is different from that available to user programs. The following format conversions are available:

```
%s      %u      %d      %o      %x      %D
```

NOTE: Operation of printf() in kernel mode occurs with elevated priority. All interrupts (even the clock) will be locked out. The system will basically be stalled until printf() completes.

The user structure is NOT used for input or output.

4.0 RANDOM CAUTIONS

This section covers various topics of interest to the writer of device drivers. Some of the information may be of interest, and some may pertain to your particular application.

4.1 USE OF THE FLOATING POINT PROCESSOR

Typically, drivers do not use the floating point processor. However, there may be an application where it is essential to use the FPP.

The ULTRIX-11 kernel does not use FPP for its own purposes. It does, however, save and restore the floating point

```
*       context as part of the process context switching procedure.
*       Therefore, it is important for the driver using the FPP to
*       save the context prior to, and restore the context after
*       using the FPP for its own arithmetic.  On systems without
*       a floating point processor which have software floating
*       point emulation within the kernel, the use of the floating
*       point instructions within kernel mode is not supported.
*
*       The driver should never attempt to suspend (sleep()) while
*       using the FPP.  To do so could cause the driver's floating
*       point context to be saved as the current user's context.
*
*  4.2  NEVER ATTEMPT TO SUSPEND PROCESSING AT INTERRUPT LEVEL
*
*       This warning has been stated earlier.  When an interrupt
*       is received, and the driver is called, all context has
*       been established on the current user's stack.  This user
*       could be anyone, including the ULTRIX-11 kernel itself, in
*       the form of the null process/scheduler.  The interrupt
*       must be processed in its entirety in order to leave the
*       context of the interrupted process intact, and to leave
*       the processor in the state it was in prior to the interrupt.
*       Therefore, the driver should NEVER attempt to sleep()
*       while at interrupt level.
*
*  4.3  PROVIDE FOR SLEEP()/WAKEUP() SYMMETRY
*
*       For each process which is suspended for an event, there is
*       a corresponding event identifier.  The identifier is
*       specified as the argument to both sleep() and wakeup().
*       It is important to bear in mind that, once the process
*       is suspended via a sleep(EVENT_ID), the only way for the
*       process to resume execution is for "someone" to provide
*       for an equivalent wakeup(EVENT_ID) at a later time.
*       Failure to provide for this symmetry could leave processes
*       suspended pending an event which will never occur.
*
*  4.4  USE OF ELEVATED PRIORITIES
*
*       Priorities greater than zero should be used only when
*       necessary.  The priority scheme is used to protect
*       context sensitive code and data from changes during
*       its use.  Misuse of processor priorities can lead to
*       severe performance problems, and strange interactions
*       between devices.
*
*  4.5  USE OF THE UNIBUS MAP
*
*       There is a clearly defined interface for use of the Unibus
*       Map for those devices which perform 18-bit I/O operations
*       on systems with 22-bit addressing.  Failure to use these
*       routines can (will) cause conflicts with other drivers in the
*       system, with data being transferred to or from the wrong
*       addresses.
```

```
*       Also, once a UMR or set of UMRs is allocated, it is important
*       that it/they be deallocated when no longer necessary for use.
*       This will allow other devices to use them, since a shortage
*       of available UMRs can cause the DMA device drivers to all
*       enter a sleep state pending the availability of UMRs.
*
*
*  5.0  DATA STRUCTURES
*
*       This section describes the data structures which are
*       related to devices on the ULTRIX-11 system.  Most of
*       these are defined in the configuration file, c.c
*       (/usr/conf/c.c), are allocated as part of sysgen,
*       and initialized as part of system startup.
*
*  5.1  DEVICE CSR (CONTROL AND STATUS REGISTER) ASSIGNMENTS
*
*       Table io_csr[] contains the CSR assignments for all
*       controllers on the system (those which are supported
*       by the software).  This table is indexed by the
*       constant XX_RMAJ (XX is the device name), which is
*       defined in /usr/include/sys/devmaj.c.  Typically, the
*       CSR address is the first address of a set of registers
*       used to control a particular device.
*
*         extern io_csr[];
*
*  5.2  UNIBUS ADDRESS EXTENSION REGISTER
*
*       Table io_bae[] contains a value which is the offset
*       from the base CSR of the 22-bit address register.
*       This table is indexed by the constant XX_BMAJ (XX is
*       the device name), which is also defined in file
*       /usr/include/sys/devmaj.c.
*       Certain devices (e.g. RL02) allow DMA transfers
*       using a 22-bit address register, as opposed to a
*       18-bit address register.  For those devices, the
*       io_bae[XX_BMAJ] entry contains the offset from the
*       base CSR address (io_csr[XX_RMAJ]) of the 22-bit
*       address extension register.  Those devices which
*       do not have 22-bit address capabilities have an
*       io_bae[XX_BMAJ] entry of zero.  By default, sysgen
*       initializes this entry to zero.  It is the driver's
*       responsibility to set the appropriate value in this
*       table.
*
*         extern char io_bae[];
*
*       Within the open() routine, code can be provided to
*       initialize the io_bae[] entry.  When the system is
*       first initialized as part of startup, all user-
*       written drivers are called via the open() entry
*       point, with a 'flag' value of -1.  At this time,
*       the driver should initialize the io_bae[] entry.
```

```
*
*  5.3   DEVICE CSR STRUCTURE DEFINITIONS
*
*        Within the driver, the format of the CSR register set
*        should be defined.  Typically, devices have several
*        different registers (contiguously on the I/O page)
*        which each serve a specific purpose.  An example of
*        this is shown below:
*
*          struct device
*            {
*               int     XXcs;      <- control and status register
*               int     XXwc;      <- word count register
*               caddr_t XXba;      <- bus address register
*               int     XXda;      <- disk address register
*               int     XXerr;     <- error register
*               int     XXbae;     <- bus address extension register
*            };
*
*  5.4   STRUCTURES FOR DRIVERS REQUIRING TERMINAL (TTY) STRUCTURES
*
*        Terminal device drivers which require tty data structures
*        should make provisions for their allocation.
*
*          extern struct tty tty_ts[];
*
*          struct tty *XX_tty[NUMBER_OF_TTYS] = {
*                      &tty_ts[1],
*                          .
*                          .
*                          .
*                      &tty_ts[NUMBER_OF_TTYS] };
*
*  5.5   LOCAL BUFFERS REQUIRING STATIC UNIBUS MAP ASSIGNMENT
*
*        Certain devices (such as the TS11 magnetic tape unit, or
*        the MSCP class of disks) require special buffers for their
*        control.  On processors where a UNIBUS MAP is provided to
*        translate the 18-bit buffer addresses setup in the CSRs
*        into real 22-bit addresses, these buffers must be placed
*        into a central area where they can be permanently assigned
*        a UNIBUS MAP register.  The sysgen program handles these
*        buffers specially, to insure that they are placed into the
*        common statically mapped area.  It is important that the
*        name follow the pattern!  The current limitation on the
*        amount of space allocated to the statically mapped area
*        is one UMR (8KB), or 8192 bytes.  Should the sum of all
*        contributions by all configured drivers exceed this limit,
*        the sysgen procedure will produce a diagnostic message,
*        then abort the sysgen.  Also, the boot program will not
*        allow the kernel to be loaded if the processor has a
*        Unibus Map, and the static area is larger than 8K-bytes.
*
*          #define XX_BUFSIZ n    <- total size of all buffers combined
```

```
*
*          union {
*                  char XX_mbs[XX_BUFSIZ]; <-- total size of buffers
*                  int  XX_mb1;            <-- REAL driver definitions
*                      .
*                      .
*                      .
*                } XX_mbuf;                <-- use the standard name!
*
*  5.6.1 CELLS IDENTIFYING AVAILABLE HARDWARE
*
*        There are several kernel variables which can be used to
*        determine the presence of certain hardware.  Here is a list
*        of the current cells available:
*
*              sepid          - if set, the CPU supports separate
*                               instruction and data spaces (this
*                               also identifies the system as one
*                               which supports all three modes:
*                               user, supervisor, and kernel).
*
*              ubmaps         - if set, the CPU has a Unibus Map.
*
*              rn_ssr3        - software copy of the MMR3 status
*                               register (this can be used to check
*                               the activation of separate I/D space
*                               for the various modes), with the
*                               current software release level in
*                               the high byte.
*
*              cputype        - integer representing the model of
*                               the PDP-11 central processor which
*                               the system is currently running on
*                               (i.e. 70 for PDP-11/70)
*
*        NOTE: To access these cells from assembler code, be sure
*              to prefix the appropriate symbol with an underscore
*              (such as _sepid, _ubmaps, _rn_ssr3, _cputype).  This
*              is a convention of c.
*
*/

/*
 * Prefix (header) files.
 *
 * It may be necessary for drivers to include the definitions of the
 * various data cells and data structures which exist within the
 * kernel.  Some of these data structures were described above, with
 * a reference made to the actual files which contain the definitions.
 * Here are some others which should be useful.
 *
 *  /usr/include/sys/param.h
 *
 *    This file contains the definitions of the ULTRIX-11 system.
```

```
 *    It also contains the "macros" which are used extensively within
 *    the kernel code to perform conversions from one unit to another.
 *    See the file for more details.
 *
 *  /usr/include/sys/systm.h
 *
 *    This file contains the descriptions and definitions of many of
 *    the cells used by the kernel code, such as the current time of
 *    day as maintained by the clock handling routines.  See the file
 *    for more details.
 *
 *  /usr/include/sys/devmaj.h
 *
 *    This file contains the device major number assignments for the
 *    current system.  It is necessary to use this file in order to
 *    obtain the major values used to index various data structures
 *    within the kernel.
 *
 */

#include <sys/param.h>
#include <sys/systm.h>
#include <sys/devmaj.h>

/*
 * Include the additional prefix (header) files as needed.  These are
 * driver dependent.  The following included files are being used as an
 * example, and may or may not be required by your specific driver.
 * Other files should be included as needed.
 */

#include <sys/buf.h>
#include <sys/tty.h>
#include <sys/dir.h>
#include <sys/user.h>

/*
 * Define a structure similar to the following to access the device's
 * hardware controller registers.  This is described in section 5.3.
 */

/*#define XX_BAEOFF 012 /* Offset from base CSR of Bus Addr Extension */
                        /* register if the device has one, otherwise */
                        /* define as ZERO. */
#define XX_BAEOFF 0

struct device
{                                   /*********************************/
        int     XXcs;               /* Control and status register   */
        int     XXwc;               /* Word count register           */
        caddr_t XXba;               /* Bus address register          */
        int     XXda;               /* Disk address register         */
        int     XXerr;              /* Error register                */
        int     XXbae;              /* Bus address extension register */
```

```
};                                  /***********************************/

/*
 * This is where the driver obtains the base CSR address for the
 * device.  The actual CSR assignment is made by the sysgen procedure,
 * and the value assigned can be obtained from cell io_csr[XX_RMAJ].
 * This is described in detail in section 5.1 (see file /sys/conf/c.c).
 */

extern int io_csr[];

/*
 * For devices which support 22-bits of direct memory addressing,
 * the symbol XX_BAEOFF should be non-zero.  The value of _BAEOFF
 * should be stored in the appropriate cell of the io_bae array,
 * as described by section 5.2.  If the device has 22-bit addressing,
 * the driver should set the value in io_bae[XX_BMAJ] when called
 * to perform the first open() operation.  By default, this cell
 * has been initialized to zero by the sysgen procedure.
 */

extern char io_bae[];

/*
 * Define the number of units/lines the device has.
 */

#define XX_NUNIT 0

/*
 * For devices which require terminal (tty) structures, the following
 * definitions should be made.  See section 5.4 for details.
 */

/*
 *  #define NUM_OF_TTYS n
 *
 *
 *  extern struct tty tty_ts[];
 *  struct tty *XX_tty[NUM_OF_TTYS] = {
 *              &tty_ts[1],
 *                   .
 *                   .
 *                   .
 *              &tty_ts[NUM_OF_TTYS]
 *             };
 *
 */

/*
 * If the device is a block device which supports raw access, then
 * the following buffer header can be used to pass to physio() to
 * convert the raw operation into a buffered operation.
 *
```

```
 */

struct  buf     XXrawbuf;

struct  buf     XXtab;  /* DO NOT REMOVE */

/*
 *      XX LOCAL UNIBUS MAPPED BUFFERS
 *
 * If the processor does not have a Unibus Map or the device
 * is not a DMA (Direct Memory Access) device, ignore but DO
 * NOT remove the following data structure (union).
 *
 * The XX_mbuf union is used if the device does DMA transfers
 * to/from a local buffer. This is most common for packet protocol
 * devices like the TS11 magnetic tape or MSCP disks. These DMA
 * transfers are not usually I/O data transfers, but transfers
 * involving device control information instead.
 *
 * See section 5.5 for details, and especially note the limitations
 * imposed upon the summed total of all device static areas.
 *
 * This particular example DOES NOT use this static area. Therefore,
 * since a length of zero is not allowed, the area here is defined as
 * containing a buffer equivalent in length to one 16-bit word.
 */

#define XX_BUFSIZ 2     /* TOTAL size of all buffers combined */
                        /* for this particular driver (2=min) */

union {
        char    XX_mbs[XX_BUFSIZ];      /* TOTAL size of buffers */
        int     XX_mb1;                 /* Driver's real definitions */
} XX_mbuf;

/*
 *      SAVE/RESTORE FLOATING POINT REGISTERS
 *
 * Floating point calculations may be performed within the
 * driver ONLY if the following rules are followed:
 *
 *      The processor must have floating point hardware.
 *      The floating point simulator cannot be used to do
 *      floating point instructions within the kernel.
 *
 *      The driver must save and restore the floating point
 *      hardware registers, see sample code in XXstrategy().
 *
 *      The floating point portion of the code must not call
 *      sleep() or be interruptable!
 *
 *      Floating point numbers cannot be printed, the in-kernel
 *      version of printf does not support floating point number
 *      formats.
```

```
 *
 * For additional details, see section 4.1.
 */

/*#define XX_FPUSED 1   /* Define if driver has floating point support */

#ifdef  XX_FPUSED

struct {
        int     XX_fps;         /* FP status register */
        double  XX_fpr[6];              /* FP registers */
} XX_fpsav;

#endif  XX_FPUSED

/*
 *      DEVICE OPEN ROUTINE
 *
 * The device open() routine is called from the kernel in order
 * for the driver to initialize the device prior to its use.
 * There are two circumstances under which the kernel calls the
 * open() routine.
 *
 *      1. open(dev, -1)
 *
 *         This call is made by kernel to explicitly allow the
 *         driver to initialize the various fields necessary
 *         to maintain the device.  The driver is called as part
 *         of system startup.  For certain devices, this is aimed
 *         at initializing the io_bae[XX_BMAJ] cell, and verifying
 *         that the device hardware (controller) is present, and
 *         operational.  The driver may also malloc() user memory,
 *         or preallocate blocks from the buffer cache if necessary.
 *         Device initialization occurs after the clists, buffer
 *         cache and inodes have been initialized, and the system
 *         clock has been started and root file system opened and
 *         mounted.
 *
 *      2. open(dev, flag)
 *
 *         This call is made by the kernel as part of an open(2)
 *         system call.  Return values (status) can be returned
 *         to the requestor via cell u.u_error.
 *
 * Under both circumstances, the argument 'dev' contains the device's
 * major/minor number ((XX_BMAJ << 8)|minor), but for the open(,-1),
 * the minor number will always be zero.  (For open(,flag), the device's
 * major/minor number could also be ((XX_RMAJ << 8)|minor).)
 * Argument 'flag' is used to indicate whether the device should be
 * opened for read-only, write-only, or read/write access.
 */

/*
 * This cell is used locally by the driver to determine if the
```

```
 * controller is present and/or operational.  Initialized to zero,
 * a value of 1 is indicative of the device not being usable.
 */

int     XX_dead;

/*
 * This cell is used locally by the driver to determine if the
 * controller for the device is active (processing a request) or not.
 * Initialized to zero, a value of 1 is indicative of the device being
 * busy (as far as the software is concerned).  This is particularly
 * useful for handling stray interrupts, since a device should not
 * interrupt unless it is the completion of an operation.
 *
 * NOTE: A stray interrupt can occur under the following two
 *       circumstances:
 *
 *             1. Multiport disk and tape controllers will interrupt
 *                if the requested unit becomes available after having
 *                returned an error when the driver attempted to
 *                request the drive.
 *             2. The RK07 disk controller will interrupt when the
 *                device is physically spun down.
 */

int     XX_active;

XXopen(dev, flag)
{
        register struct device *XXaddr;

        /*************************************************************/
        /* flag value of -1 indicates that controller initialization */
        /* is to be performed (this call is part of kernel startup)  */
        /*************************************************************

        if (flag == -1) {
                if(fuiword((caddr_t) io_csr[XX_RMAJ]) == -1) {

                        /*********************************************/
                        /* The driver is configured, but the CSR is  */
                        /* not present at the address specified in   */
                        /* io_csr[XX_RMAJ].  This is indicated by -1 */
                        /* being returned by fuiword() (see section  */
                        /* 3.5 for the note concerning the return    */
                        /* value -1).                                */
                        /*********************************************/

                        XX_dead = 1;
                        return;
                }

                /*****************************************************/
                /* Otherwise, the CSR is present on the I/O page.    */
```

```
                /* In this case, boot leaves the value originally    */
                /* defined in io_csr[XX_RMAJ] as it was.             */
                /* Now, the driver is responsible to initialize the  */
                /* bus address extension offset in table io_bae[].   */
                /*****************************************************/

                io_bae[XX_BMAJ] = XX_BAEOFF;
                return;
        }

        if (XX_dead) {

                /*****************************************************/
                /* If the device is non-operational (or just not     */
                /* present, return an error to the caller. This      */
                /* particular error indicates that the device or     */
                /* CSR is not present in the current system.         */
                /*****************************************************/

        bad:
                u.u_error = ENXIO;
                return;
        }

        /*************************************************************/
        /* Obtain a local copy of the device's CSR base address.     */
        /* If the device has multiple units, it may be necessary to  */
        /* modify the CSR address in the case where multiple CSRs    */
        /* are involved.  Note that a requirement of ULTRIX-11 with  */
        /* respect to multiple controllers of a particular type is   */
        /* that all controllers of the same type must be physically  */
        /* contiguous on the I/O page.  The io_csr[] table contains  */
        /* the address of the first such CSR of the series.          */
        /* Note that XXaddr is a structure pointer.                  */
        /*************************************************************/

        XXaddr = io_csr[XX_RMAJ];

        /*************************************************************/
        /* Here the driver validates the unit number.  The minor     */
        /* number is the unit of the particular device on the        */
        /* system.  The symbol XX_NUNIT has been defined as the      */
        /* number of units for this device.  Note that unit number   */
        /* validation is device specific.                            */
        /*************************************************************/

        if (minor(dev) >= XX_NUNIT)
                goto bad;
}

/*
 *      DEVICE CLOSE ROUTINE
 *
 * The device close() routine is called from the kernel in order for
```

```
 * the driver to "close off" access to a particular device.  This
 * routine is called as part of the close(2) system call processing,
 * however, it is NOT called for each close() issued by all processes.
 * Rather, it is called when the LAST accessor of the device issues
 * the close(2) system call.
 *
 *      close(dev, flag)
 *
 * Argument 'dev' is device major/minor number ((XX_RMAJ << 8)|minor).
 * Argument 'flag' is similar to argument 'flag' used for the open(2)
 * system call/routine, as discussed previously, and pertains to the
 * 'flag' specified by the last accessor of the device.
 *
 * The close() routine may have several uses:  close() processing
 * is device specific.  For terminal devices, close() can be used to set
 * the terminal device to a predefined set of characteristics.  For
 * disks, the device could be spun down (for example, the MSCP/RA
 * class of disks allow the drive to be spun down or up on command).
 *
 */

XXclose(dev, flag)
{
}

/*
 *      DEVICE STRATEGY ROUTINE
 *
 * The device strategy() routine is called by the kernel to perform I/O
 * operations for block-oriented devices (only devices such as disks or
 * tapes use the strategy routine (see section 2.2 for details)).  The
 * argument passed to the routine is the address of a control block
 * which basically acts as an argument list for the block device driver.
 * It contains fields necessary to relay information pertaining to the
 * block number to be read or written, the buffer address and length,
 * the operation code to be performed, and a field reserved to return
 * the status of the I/O operation (see section 2.2.2 for details).
 *
 *      strategy(bp)
 *
 * See section 2.2.2 for a description of the buffer pointer (bp) and
 * the contents of the buffer descriptor.  Section 2.2.3 (and the
 * associated information in section 3.9) contains information about
 * the use of the Unibus Map, for the use of DMA (NPR) devices on
 * 22-bit systems.
 */

XXstrategy(bp)
register struct buf *bp;
{

/*
 * Throughout this routine are sample sections of code which demonstrate
 * the use of the floating point processor within a device driver.
```

```
 * While the code is conditionalized on the symbol XX_FPUSED, it can be
 * removed in its entirety if desired.
 *
 * See section 4.1 for a warning concerning the use of the floating
 * point processor.
 */

#ifdef  XX_FPUSED

        int     XX_pri;
        double  XX_f1, XX_f2;

#endif  XX_FPUSED


        if (!io_bae[XX_BMAJ])

                /*******************************************************/
                /* If the device does not have direct 22-bit address */
                /* capability, the io_bae[] entry will contain a     */
                /* zero, since open(,-1) did not place an offset for */
                /* a 22-bit address extension register in the set of */
                /* CSRs for this device.                             */
                /*******************************************************/

                mapalloc(bp);

#ifdef  XX_FPUSED

        /*************************************************************/
        /* This sample floating point code demonstrates how device  */
        /* drivers could use the FPP.  Note that the code executes  */
        /* at elevated priority and that while non-interruptable    */
        /* it saves the current FPP context, uses the FPP for its   */
        /* own purposes, then restores the saved context.  Finally, */
        /* it returns the processor to the priority it was at prior */
        /* to using the floating point processor.                    */
        /*************************************************************/

        XX_pri = spl7();
        savfp(&XX_fpsav);
        XX_f1 = 1.43;
        XX_f2 = XX_f1 * 345.567;
        restfp(&XX_fpsav);
        splx(XX_pri);

#endif  XX_FPUSED

}

/*
 *      DEVICE START ROUTINE
 *
 * The device start() routine is NOT called from the kernel.  It is
```

```
 * an internally used routine (internal to the driver) which typically
 * initiates an I/O transfer.  This particular routine is setup for
 * use with a terminal device, which has tty (terminal) data structures.
 */

XXstart(tp)
register struct tty *tp;
{
        register struct device *XXaddr;

        XXaddr = io_csr[XX_RMAJ];
        /* may need to add the unit number depending on device type */
}

/*
 * If the device has only one vector, use
 * XXrint and leave XXxint as it is.
 * If the device interrupts at a br level other
 * than 5 edit l.s to change the br level.
 */

/*
 *      DEVICE INTERRUPT ROUTINES
 *
 * The device interrupt routines are used as part of driver processing.
 * (See section 1.3 for a discussion of interrupt processing.)  If the
 * device is a character device, in MOST cases, it will have two
 * interrupt vectors, one for input (reader) interrupts, and one for
 * output (transmitter) interrupts.  On the other hand, block devices
 * typically have only one interrupt vector.  These two routines are
 * called XXrintr (reader interrupts) and XXxintr (transmitter
 * interrupts).  Devices which have a single interrupt vector should
 * use only the XXrintr routine.  (See sections 1.3 and 4.2 for notes
 * and warnings concerning device processing while at interrupt level.)
 */

/*
 *      DEVICE READER INTERRUPT ROUTINE
 */

XXrint(dev)
{

        if (XX_active == 0) {

                /*******************************************************/
                /* If an interrupt occurs and the device currently     */
                /* is not performing an operation we requested, then   */
                /* the situation can be logged as a stray interrupt    */
                /* as follows.                                         */
                /*******************************************************/

                logsi(io_csr[XX_RMAJ]);
                return;
```

```
        }

/*
 * If the vector is not currently being used for a particular
 * controller, then the interrupt should be logged in the following
 * fashion:
 *
 *      logsi(vector);
 *      return;
 *
 * Where (vector) is the device's interrupt vector address.
 */

        /*********************************************************/
        /* Typically the I/O operation completion will be signaled */
        /* by the hardware as an interrupt.  If this is the case,  */
        /* then for block devices the operation should be completed */
        /* using the iodone() routine (see section 3.7.8 for the   */
        /* details of buffered I/O completion).                    */
        /*********************************************************/

        iodone(bp);

}

/*
 *      DEVICE TRANSMITTER INTERRUPT ROUTINE
 */

XXxint(dev)
{

        if (XX_active == 0) {

                /*****************************************************/
                /* If an interrupt occurs and the device currently   */
                /* is not performing an operation we requested, then */
                /* the situation can be logged as a stray interrupt  */
                /* as follows.                                       */
                /*****************************************************/

                logsi(io_csr[XX_RMAJ]);
                return;
        }

/*
 * If the vector is not currently being used for a particular
 * controller, then the interrupt should be logged in the following
 * fashion:
 *
 *      logsi(vector+04);
 *      return;
 *
 * Where (vector) is the device's interrupt vector address.
```

```
 */
}

/*
 *      DEVICE READ AND WRITE ROUTINES
 *
 * Devices which are character oriented are often capable of full
 * duplex operation.  This means that the device can have both
 * input and output requests active at the same time.  This is
 * evident by the fact that these devices also have two separate
 * interrupt routines -- one for input and one for output.
 * This is unlike the block oriented devices which can have
 * only one operation in progress per controller (seek operations
 * for multi-unit disk controllers are overlappable, but the data
 * transfers are not).
 *
 */

/*
 *      DEVICE READ ROUTINE
 *
 * The read routine is called from the kernel as part of the read(2)
 * system call for character devices.  The interface is discussed in
 * section 2.1, with the routines discussed in section 3.6 and 3.8
 * being extremely helpful.
 *
 * If the driver supports raw mode access, then the driver will be
 * called here, instead of the strategy routine.  Physio is used to
 * convert the raw transfer into a buffered transfer.  See section
 * 3.7A for details.
 */


XXread(dev)
{
        physio(&XXstrategy, &XXrawbuf, dev, B_READ);
}

/*
 *      DEVICE WRITE ROUTINE
 *
 * The write routine is called from the kernel as part of the write(2)
 * sytem call for character devices.  The interface is discussed in
 * section 2.1, with the routines discussed in section 3.6 and 3.8
 * being extremely helpful.
 *
 * If the driver supports raw mode access, then the driver will be
 * called here, instead of the strategy routine.  Physio is used to
 * convert the raw transfer into a buffered transfer.  See section
 * 3.7A for details.
 */

XXwrite(dev)
{
```

```
        physio(&XXstrategy, &XXrawbuf, dev, B_WRITE);
}

/*
 *      DEVICE CONTROL ROUTINE
 *
 * Many devices require a special mechanism to be able to perform
 * operations which are not data transfers.  These "functions" enhance
 * the useful of the device, and often enable the user to alter the
 * behavior of the device or its processing by the kernel.
 *
 * The device ioctl() routine is called from the kernel as part of the
 * ioctl(2) system call.
 *
 *      ioctl(dev, cmd, addr, flag)
 *
 * Argument 'dev' contains device major/minor number.  Argument 'cmd'
 * contains the function code of the operation to be performed (see file
 * /usr/include/sys/ioctl.h for the function codes).  Argument 'addr'
 * is function specific.  Typically, it contains the address of a
 * parameter list.  This list can be used for either input or output,
 * depending upon the function code.  Argument 'flag' is the third
 * parameter of ioctl(), and is device specific.
 *
 */

XXioctl(dev, cmd, addr, flag)
caddr_t addr;
{
}

/*
 *      DEVICE STOP ROUTINE
 *
 * The device stop() routine is called from the kernel (terminal
 * driver) to stop output when an XOFF (control/S) is received.
 * Typically, this involves the clearing of the "transmitter
 * interrupt enable" bit.  This would cause the device to stop
 * interrupting when it is ready to transmit another character
 * to the terminal.  As a result, output will stop until the
 * transmitter interrupt is again enabled.
 *
 * In some cases, the stop routine may not be necessary.  For
 * example, the DZ[V]11 driver uses a status bit in the terminal
 * (tty) structures which indicates that output should be stopped.
 * When the transmitter interrupts to receive another character,
 * the driver will not load another character, and it will clear
 * the "transmitter interrupt" bit, thereby stopping output.
 *
 * When XON is received by the terminal driver, it will clear the
 * status bit and call the device start() routine.  As a result,
 * output will resume on the device.
 *
 */
```

```
XXstop(tp, flag)
register struct tty *tp;
{
}
```