Opus 100PM User Manual

800-00237-000

Opus Systems
20863 Stevens Creek Blvd
Building 400
Cupertino, CA 95014

(408) 446-2110

Trademarks

UNIX is a trademark of AT&T Bell Laboratories

FEDERAL COMMUNICATIONS COMMISSION
RADIO FREQUENCY INTERFERENCE STATEMENT

The Opus UNIX processor in the PC generates and uses radio frequency energy and if not installed and used properly, that is, in strict accordance with the instructions contained in this manual, may cause interference to radio and television reception. It has been type tested and found to comply with the limits for a Class B computing device in accordance with the specifications of Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference in a residential installation. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause interference to radio or television reception, which can be determined by turning the host PC on and off, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient the receiving antenna.
- Relocate the computer with respect to the receiver.
- Move the computer away from the receiver.
- Plug the computer into a different outlet so that computer and receiver are on different branch circuits.
- Ensure that all PC card mounting screws, attachment connector screws, and ground wires are tightly secured.
- Ensure that the PC card slot covers are in place when no card is installed.
- Use shielded, grounded cables with in-line filters, if necessary.

If necessary, consult an experienced radio/television technician for additional suggestions. The booklet *How to Identify and Resolve Radio-TV Interference Problems*, prepared by the Federal Communications Commission, may also be helpful. This booklet is available from the US Government Printing Office, Washington, DC 20402, Stock No. 004-000-00345-4.

REVISION HISTORY

Publication Number	Date	Changes
800-00237-000	February 1987	First edition; formerly titled *Opus532 User Manual*

CONTENTS

1 OVERVIEW

1.1 General Description 1-2
1.1.1 Opus5 Software 1-2
1.1.2 UNIX Processor 1-3
1.2 Features 1-3
1.3 Specifications 1-4

2 HARDWARE INSTALLATION

2.1 Verifying Factory-Selected Settings 2-1
2.1.1 Opus 108PM Factory Settings 2-1
2.1.2 Opus 110PM Factory Settings 2-4
2.2 Attaching Opus161 or 321 Memory Expansion Board 2-9
2.3 Adding Opus323 Memory Expansion Board 2-10
2.4 Installing Opus 100PM UNIX Processor 2-10

3 Opus5 SOFTWARE INSTALLATION

3.1 Determining Required Size of System Partitions 3-3
3.2 Establishing DOS Partition 3-4
3.2.1 Backing Up DOS 3-5
3.2.2 Altering Size of DOS Partition 3-5
3.2.3 Restoring DOS 3-6
3.3 Running **opinit** 3-7
3.3.1 Running **opconfig** for System Initialization 3-8
3.4 Running **opload** 3-13

4 Opus5 ADMINISTRATIVE PROCEDURES

4.1 Day-to-day Start-Up 4-2
4.2 Multiuser Mode 4-3
4.3 Setting Time Zone 4-5
4.3.1 Daylight Savings Time 4-7
4.4 The **.profile** File 4-9
4.4.1 Multiple Users with Assorted Terminals 4-11
4.5 Disabling Automatic Crontab Activity 4-12
4.6 Assigning Passwords 4-13
4.7 Adding Users 4-14
4.7.1 Adding a Line to **/etc/passwd** 4-15
4.7.2 Creating a Home Directory 4-16
4.7.3 Giving User a **.profile** 4-17
4.7.4 Changing Ownership of User's Home Directory 4-17
4.8 Using **terminfo** 4-17
4.9 Setting Up UUCP 4-19
4.9.1 Hardware Considerations 4-19
4.9.2 Software Checklist 4-20
4.10 Adding a Terminal 4-23
4.10.1 Modifying **/etc/gettydefs** and Using **stty** 4-25
4.11 Connecting a Serial Printer 4-28
4.12 Setting Up the Line Printer Spooler 4-31
4.13 Changing the Maximum File Size 4-33
4.14 Backup and Restore 4-35
4.14.1 Backing Up Opus Release Diskettes 4-35
4.14.2 Backing Up Opus5 File Systems and Files 4-35
4.15 Shutting Down Opus5 4-38

5 OPUS ENHANCEMENTS TO SYSTEM V

5.1 Shell-Level Interface to DOS 5-1
5.1.1 DOS Commands from Opus5 5-2
5.1.2 Opus5 Commands from DOS 5-4

5.1.3 Transferring Files between UNIX and DOS 5-8
5.1.4 DOS Return Code 5-9
5.2 Opus Library 5-10
5.2.1 Notes for Using Opus Library 5-11
5.3 Local Terminals 5-12
5.4 Line Printer Driver 5-16
5.5 Enhanced **dial**(3C) Function 5-17
5.6 VDI Interface 5-17
5.6.1 Setting Up VDI 5-18
5.6.2 Installing VDI 5-18
5.6.3 VDI Usage Notes 5-19
5.7 CGI Interface 5-21
5.7.1 Setting Up CGI 5-21
5.7.2 Installing CGI 5-21
5.7.3 CGI Usage Notes 5-22
5.8 The /**unix** Command 5-23
5.9 Sessions 5-23
5.9.1 Sessions — Usage Hints and Examples 5-27
5.10 Using Opdisk for Large-Capacity Disks 5-29
5.10.1 Get a Priam Disk 5-30
5.10.2 Physically Install the Priam Disk 5-30
5.10.3 Run AT SETUP 5-31
5.10.4 Create DOS Partition 5-33
5.10.5 Modify **config.sys** 5-34
5.10.6 Modify DOS System Files 5-35
5.10.7 Install **opdisk** 5-35
5.10.8 Install UNIX 5-36

6 DEVICES

6.1 Console Driver 6-4
6.1.1 **console** 6-4
6.1.2 **doscon** 6-4
6.2 DK Driver 6-5
6.2.1 c:, d:, ... Devices 6-6

6.2.2 c:xt, d:xt, ... Devices 6-8
6.2.3 fil# Devices 6-8
6.2.4 a:bios, b:bios, ... Devices 6-9
6.3 SYS Driver 6-9
6.4 MEM Driver 6-10
6.5 ERR Driver 6-10
6.6 Serial Driver 6-10
6.7 LP Driver 6-13
6.8 PRF Driver 6-14
6.9 SXT Driver 6-14
6.10 DOS Driver 6-14
6.11 Clock Driver 6-15
6.12 Tape Driver 6-16
6.13 Ethernet Driver 6-18
6.14 PC Net BIOS 6-19
6.15 GN Driver 6-19
6.16 VDI Driver 6-21

7 SYSTEM MESSAGES

7.1 Level 1 and Level 2 Tests 7-1
7.1.1 Level 1 Tests 7-1
7.1.2 Level 2 Tests 7-5
7.2 Messages 7-7
7.3 I/O Error Messages 7-20
7.3.1 Error Message Format 7-20
7.3.2 How to Interpret the Dix 7-22
7.3.3 Device Error Messages Grouped by Device 7-24
7.3 Turning On Opmon Error Reporting 7-39

8 Opus 100PM HARDWARE THEORY OF OPERATION

8.1 Overview 8-1
8.2 Hardware Configuration Options 8-2
8.2.1 Software Option 8-4
8.2.2 Interrupt Request Level 8-4
8.2.3 Segment Number 8-5

8.2.4 Number of Wait States 8-7
8.3 Operation.....8-8
8.4 Opus 100 I/O.....8-10
8.5 PC I/O.....8-12

APPENDIX A: Opus5 SOFTWARE INSTALLATION USING DOS FILES

APPENDIX B: UNIX vs DOS — HELPFUL HINTS

APPENDIX C: FILE GROUPS IN THE C3 RELEASE

APPENDIX D: HINTS FOR PORTING FORTRAN 77 PROGRAMS

APPENDIX E: ADDING SERIAL PORTS

APPENDIX F: Opus5 REFERENCE MANUAL PAGES

dos(1*)
ducp(1*)
opconfig(1*)
opdos(1*)
opload(1*)
opmon(1*)
opsash(1*)
opunix(1*)
swap(1M)
tape(1*)
udcp(1*)
udiodaemon(1*)
unix(1*)
dial(3C*)
doschdir(3*)
doschdrv(3*)
doschmod(3*)
dosclose(3*)
doscreat(3*)
dosdiskfree(3*)

dosioctl(3*)
doslseek(3*)
dosmkdir(3*)
dosopen(3*)
dosread(3*)
dosrename(3*)
dosrmdir(3*)
dosunlink(3*)
doswrite(3*)
draindisk(3*)
getdiskname(3*)
getdiskparm(3*)
setdiskname(3*)
syscmdline(3*)
sysexit(3*)
syspause(3*)
sysreadio(3*)
sysreadmem(3*)
syswriteio(3*)
syswritemem(3*)
opus.cfg(4*)

1. OVERVIEW

This manual describes the Opus 100PM (Personal Mainframe) UNIX processor subsystem for the IBM PC or plug-compatible computer. The Opus 100 PM includes the Opus5 operating system and the Opus coprocessor. The Opus5 operating system is a complete port of AT&T UNIX System V (Release 2.0 Version 2) for the National Semiconductor 32016 (Opus 108PM) and 32032 (Opus 110PM) processors.

This manual covers specifications, installation, configuration, and theory of operation. It has the following chapters:

1. Overview
2. Hardware Installation
3. Opus5 Software Installation
4. Opus5 Administrative Procedures
5. Opus Enhancements to System V
6. Devices
7. System Messages
8. Hardware Theory of Operation

Appendix A: Opus5 Software Installation Using DOS Files
Appendix B: Opus5 vs DOS — Helpful Hints
Appendix C: File Groups in the C3 Release
Appendix D: Hints for Porting FORTRAN 77 Programs
Appendix E: Adding Serial Ports
Appendix F: Opus5 Reference Manual Pages

This manual is meant to be used in conjunction with the standard AT&T documentation for System V UNIX (Release 2.0, Version 2). Where relevant, use the manuals for DEC processors. You should have all the following AT&T manuals:

- *User Reference Manual* — DEC
- *User Guide*
- *Programmer Reference Manual* — DEC
- *Programming Guide*

- *Administrator Reference Manual* — DEC
- *Administrator Guide* — DEC
- *Support Tools Guide*
- *Operator Guide* — DEC

If you are doing program development on the Opus 100PM, you might also need the following manual from National Semiconductor:

- *Series 32000 Instruction Set Reference Manual* Customer Order No. NSP-INST-REF-M, NSC Publication Number 420010099-00 1B.

Throughout this manual, Opus manual pages are designated by (n*) to distinguish them from AT&T reference manual pages. "n" is the section number of the appropriate *UNIX System V Release 2.0 Reference Manual* where the Opus manual page would fit, and "*" indicates that it is an Opus command. For example, **dos**(1*) means the Opus manual page for the "dos" user command (section 1).

1.1 General Description

The Opus 100PM system consists of Opus5 software and the UNIX processor.

1.1.1 Opus5 Software

The Opus5 operating system is a full implementation of AT&T UNIX System V (Release 2.0, Version 2), with the following exceptions:

- Devices and programs for devices specific to DEC hardware, e.g., VPM subsystem, **rp** devices, **kasb**(1).
- Programs for which no source is provided with the AT&T VAX release, e.g., **/usr/games/jotto**.
- The UNIX graphics subsystem.
- The programs **crash**(1M), **config**(1M), and **sysdef**(1M), which will be in a future Opus release.

1.1.2 UNIX Processor

The Opus UNIX processor is a single-slot card that allows an IBM PC or plug-compatible computer to run the Opus5 UNIX operating system.

The Opus 108PM design includes:

- 32016 processor (CPU)
- 32082 memory management unit (MMU)
- 32081 floating point unit (FPU)
- 32201 timing control unit (TCU)

The Opus 110PM design includes:

- 32032 processor (CPU)
- 32082 memory management unit (MMU)
- 32081 floating point unit (FPU)
- 32201 timing control unit (TCU)

Both the Opus 108PM and Opus 110PM carry 1 Mbyte of memory. For the Opus 108PM, an additional 1 Mbyte of memory is available on a piggyback board (the Opus161 memory expansion board). For the Opus 110PM, an additional 1 Mbyte of memory is available on a piggyback board (the Opus321 memory expansion board), or an additional 3 Mbytes of memory are available on an expansion board (the Opus323 memory expansion board).

1.2 Features

Fast Floating Point — Hardware implementation of fast floating point arithmetic supports proposed IEEE standard for binary floating point operations.

Virtual Memory Management — Hardware support for demand-paged virtual memory management along with comprehensive software debugging features.

Onboard RAM Memory — Up to 2M bytes of onboard memory on the Opus 108PM, up to 4 Mbytes on the Opus 110PM.

Compatible — Plugs into the IBM PC backplane with no additional connections required.

Standard — Supports the Opus5 operating system, a fully implemented port of standard AT&T UNIX System V (Release 2.0, Version 2).

1.3 Specifications

Available Hardware Configurations

Processor speeds .....8 or 10 MHz
Memory
- Opus 100PM UNIX Processor.....1 Mbyte
- Opus161 Memory Expansion Board.....1 Mbyte
- Opus321 Memory Expansion Board.....1 Mbyte
- Opus323 Memory Expansion Board .....3 Mbytes
- Combined Opus 108PM and Opus161 .....2 Mbytes
- Combined Opus 110PM and Opus321 .....2 Mbytes
- Combined Opus 110PM and Opus323 .....4 Mbytes

System Requirements

DOS (2.x or 3.x) and a suitable host system, for example:
- IBM — XT, AT, or PC with expansion chassis and hard disk
- TI — Professional Computer or Business Pro
- Compaq — Plus, Deskpro, or Deskpro-286
- AT&T — 6300 with hard disk
- ITT — Xtra
- H-P — Vectra
- Sperry — PC-IT
- Tandon 286
- Zenith Data Systems
- Alternative Systems
- PC Limited
- MAD System 286
- Federal Data XL
- Tandy 1200, 3000, 3000HD

Any system with Phoenix BIOS

Slots Required

Opus 100PM coprocessor.....One long slot
Opus 100PM with 1 Mbyte
memory expansion board.....One long slot
Opus 100PM with 3 Mbyte
memory expansion board .....Two long slots

Power Requirements

Typical
Opus 108PM .....2.7 amp @ +5 vdc
Opus 110PM .....3.0 amp @ +5 vdc
Maximum
Opus 108PM .....3.0 amp @ +5 vdc
Opus 110PM .....3.8 amp @ +5 vdc
Minimum Voltage .....4.75 vdc

Environmental Requirements

Operating temperature .....0C to 35C ambient max
Storage temperature.....-40C to 85C ambient max
Relative humidity.....0 to 95% noncondensing

Dimensions

Standard IBM PC long-slot format (excluding bracket):

Length .....13.13 in. (333.5 mm)
Height.....4.20 in. (106.7 mm)
Width
Opus 110PM or Opus 108PM only .....0.45 in. (11.4 mm)
Opus 108PM with Opus161.....0.70 in. (17.8 mm)
Opus 110PM with Opus321.....0.70 in. (17.8 mm)
Opus 110PM with Opus323
front mount .....1.4 in. (35.5 mm)
rear mount.....1.1 in. (27.9 mm)

2. HARDWARE INSTALLATION

NOTE: The IBM PC address space ranges from 00000h to FFFFFh (1M byte). The Opus 100PM occupies 64K bytes and can be placed on any 64K boundary in this address space. The default installation for the Opus 100PM is at address range A0000-AFFFF (that is, segment number A).

If your system has no known conflicts at this address, proceed with the installation instructions. If your system does have a known conflict at this address, or has been modified to include hardware that uses segment number A, you must alter the default switch settings on the Opus 100PM UNIX processor. Change the Opus 100PM segment number by following the procedure described in Section 8.2.3 (Segment Number).

2.1 Verifying Factory-Selected Settings

Before installing the Opus 100PM UNIX processor board in your system unit, verify the following settings on the board. Default settings appropriate for your site have been selected at the factory; you only need to verify that the settings have not been altered during shipping and handling. For technical details of these options, see Chapter 8 — Hardware Theory of Operation.

Note that settings differ for Opus 108PM and Opus 110PM systems, and also that settings differ within Opus 108 revisions.

2.1.1 Opus 108PM Factory Settings

This section describes Opus 108PM factory settings. Opus 110PM users see Section 2.1.2.

Verify Switch Settings — At location D25 on the Opus 108PM board, find the 8-switch DIP switchpack. Verify that the switch settings are as shown in Figure 2-1.

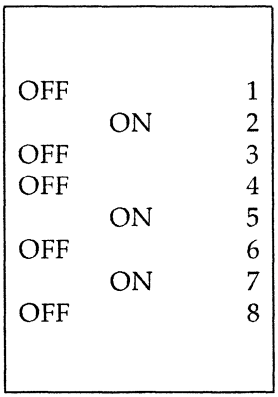


Figure 2-1. D25 Switch Settings, Opus 108PM

Use a pencil or ball-point pen to correct any switches that are not properly set.

Determine Clock Rate — To verify the factory setting for this option, you must know the clock rate of your Opus 108PM board (8 or 10 MHz). If you already know the clock rate that applies to your board, skip the next paragraph. If you do not know or are unsure which clock rate applies to your board, read the following paragraph.

At location B15 on the Opus 108PM board, find the oscillator. Read the legend printed on this component. The legend includes the speed of the oscillator (16 or 20 MHz). Divide this number by 2 to determine the clock rate of your Opus 108PM (for example, 16-MHz oscillator = 8-MHz clock rate).

Verify Jumpering — The wait jumpering differs according to different revisions of the Opus 108PM board. If your board is revision A0 - C3, the wait jumpers are at location B16. If your board is revision D0 or later, the wait jumpers are at location

B25.

To determine the revision level of your Opus 108PM board, find a line near location B15 reading *32.16 REV xx* or *Opus16 Rev xx*. "xx" is the revision level of the board.

For A0-C3 systems, verify that the two push-on jumpers are installed on the posts to look like Figure 2-2 below:

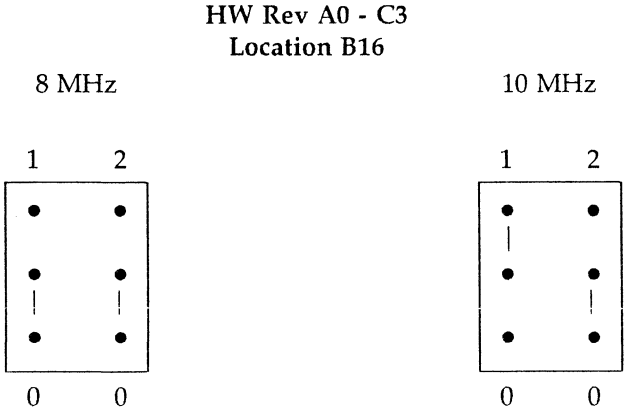


Figure 2-2. B16 Jumpers, 8- and 10-MHz, Opus 108PM

For D level boards, verify that two push-on jumpers are installed on the posts to look like Figure 2-3:

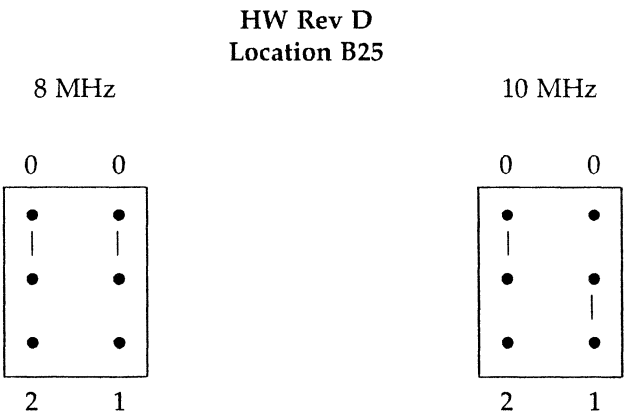


Figure 2-3. B25 Jumpers, 8- and 10-MHz, Opus 108PM

If either push-on jumper is missing or improperly installed, you must replace or reinstall the jumper before installing the Opus 108PM board in your chassis.

2.1.2 Opus 110PM Factory Settings

This section describes Opus 110PM factory settings. Opus 108PM users see Section 2.1.1.

Verify Switch Settings — At location D2 on the Opus 110PM board, find the 10-position DIP switchpack. Verify that the switch settings are as shown in Figure 2-4.

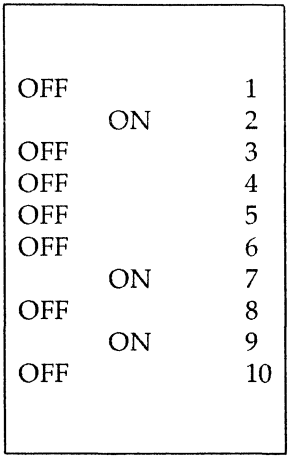


Figure 2-4. D2 Switch Settings, Opus 110PM

Use a pencil or ball-point pen to correct any switches that are not properly set.

Determine Clock Rate — To verify the factory setting for this option, you must know the clock rate of your Opus 110PM board (8 or 10 MHz). If you already know the clock rate that applies to your board, skip the next paragraph. If you do not know or are unsure which clock rate applies to your board, read the following paragraph.

At location D8 on the Opus 110PM board, find the oscillator. Read the legend printed on this component. The legend includes the speed of the oscillator (16 or 20 MHz). Divide this number by 2 to determine the clock rate of your Opus 110PM (for example, 16-MHz oscillator = 8-MHz clock rate). Make a note of this number, because you will need it to read the correct wait jumpering.

Verify Jumpering — At location H3 on the Opus 110PM board, find the three jumper blocks labeled JP1, JP2, and JP3. These

control the following functions:

Jumper Block	*Function*
JP1	Refresh (REF)
JP2	Wait States (WAIT)
JP3	Type of RAM controller (84)

32.32 REF Jumper

Because the Opus board uses dynamic RAM, all locations in the memory must be refreshed periodically to maintain the integrity of the data inside the RAMs. Most PCs provide a refresh signal from the backplane for every slot. On these PCs, the Opus board can go in any slot, and you don't have to worry about the origin of the refresh signal. However, some PCs, such as TI PCs, provide refresh for only one slot from the backplane. On PCs of this type, the Opus board cannot always be placed in a slot that receives the refresh signal, and special arrangements must be made to send refresh to the Opus board.

Jumper block JP1 controls the source of the refresh signal: either external ("EXT") or internal ("INT"). EXT tells the system to get the refresh signal "externally", from the backplane; this is usually used when the PC provides refresh for every slot. INT tells the system to get the refresh signal "internally", by a special Opus-supplied part at location F4. This is used for systems in which refresh is not available for every slot, though it works on all machines. See Figure 2-5.

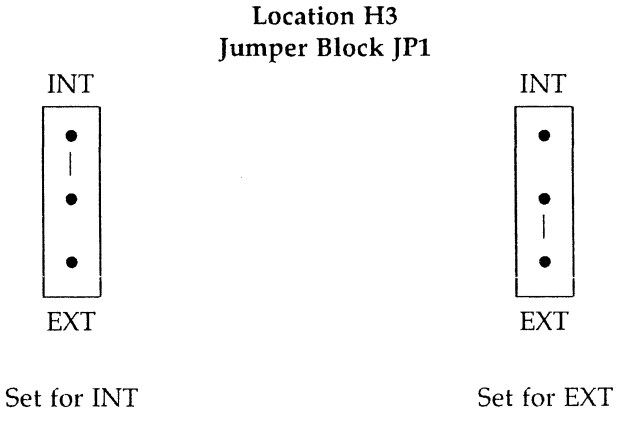


Figure 2-5. REF Jumper, Opus 110PM

32.32 WAIT Jumper

The number of wait states is controlled by the JP2 jumper block at location H3. In the 32.32 system, the factory settings provide no wait states for both 8- and 10-MHx systems, as shown in Figure 2-6.

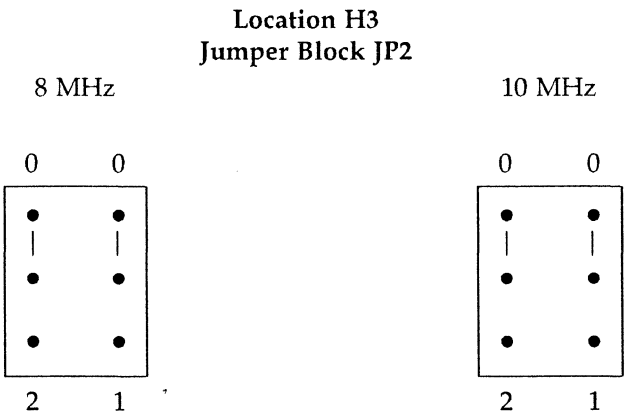


Figure 2-6. WAIT Jumpers, 8- and 10-MHz, Opus 110PM

If either push-on jumper is missing or improperly installed, you must replace or reinstall the jumper before installing the Opus 110PM board in your chassis.

Opus 110PM RAM Controller Jumper

The JP3 jumper block at location H3 on the Opus 110PM selects the type of RAM controller — either an 8409 or an 8419 part. The RAM controller is installed at the Opus factory at location F8, and the switch is set accordingly. Inspect the chip at F8 and ensure that the JP3 jumper block is set to match it. See Figure 2-7.

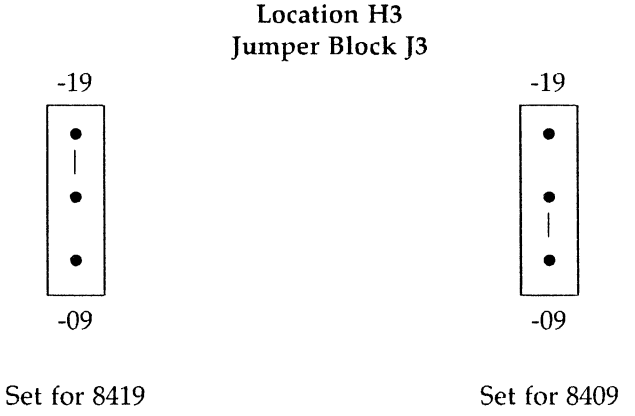


Figure 2-7. RAM Controller Jumper, Opus 110PM

2.2 Attaching Opus161 or Opus 321 Memory Expansion Board

Follow this procedure only if adding the Opus161 or Opus 321 memory expansion board as an upgrade or replacement. Otherwise, skip to Section 2.4 (Installing Opus 100PM UNIX Processor). An Opus161 memory expansion board can be added to an Opus 108PM system only, while an Opus 321 memory expansion board can be added to an Opus 110PM system only.

The Opus161 or 321 memory expansion board attaches to the Opus 100PM processor board by a pair of 25- or 40-pin connectors and is held in place by four standoffs attached to the memory expansion and screwed into place from the solder side of the Opus 100PM.

The Opus161 or 321 attaches directly to the memory array on the Opus 100PM card. The Opus161 uses two 25-pin connectors, while the Opus 321 uses two 40-pin connectors. The two connectors and the four standoffs are keyed such that after proper installation, three edges of the Opus161 or 321 board should align with three edges of the Opus 100PM board.

Tools required:

- One small screwdriver.

1. Taking care not to bend any pins, attach the Opus161 or 321 to the Opus 100PM by inserting the pins into the connectors on the Opus 100PM.
2. Verify that three edges of the Opus161 or 321 now align with three edges of the Opus 100PM.
3. From the solder side of the Opus 100PM, insert the four screws provided with the Opus161 or 321 into the four standoffs. Use the screwdriver in a clockwise motion to tighten the screws.

2.3 Adding Opus323 Memory Expansion Board

An Opus323 memory expansion board carrying 3 Mbytes may be added to any Opus 110PM board with 1 or 2 Mbytes of memory. 1-Mbyte boards become 4-Mbyte boards; 2-Mbyte boards also become 4-Mbyte boards, because the Opus 321 is removed when the Opus323 is added.

Adding an Opus323 is a factory upgrade only. Please contact Opus Systems to make arrangements.

2.4 Installing Opus 100PM UNIX Processor

The Opus 100PM UNIX processor installs in either one or two full-length slots in the PC system unit or expansion unit. With an Opus161 or 321 memory expansion board attached, the Opus 100PM coprocessor board occupies only one slot.

With an Opus323 memory expansion board attached, the Opus 110PM UNIX processor occupies either one or two slots,

depending on the system type and how the memory board is installed. An Opus 110PM with an Opus323 memory expansion board can occupy one slot when all the following conditions are met:

- it is installed in an AT;
- the Opus 323 is *rear-mounted*;
- slot #1 is used.

Otherwise, an Opus 110PM with an Opus323 occupies two slots.

Follow this procedure to install either the Opus 108PM or Opus 110PM processor. These instructions and illustrations apply specifically to an Opus 108PM in the IBM PC XT. Any other IBM-compatible expansion chassis may require different installation techniques and/or tools of different sizes.

Tools required:

- 1/4-in. and 3/16-in. hex nutdrivers
  or
- One medium-size, flat-blade screwdriver

1. Set the system power switch on the system unit to OFF. See Figure 2-8.
2. Set the power switch on the expansion unit (if any) to OFF. See Figure 2-8.

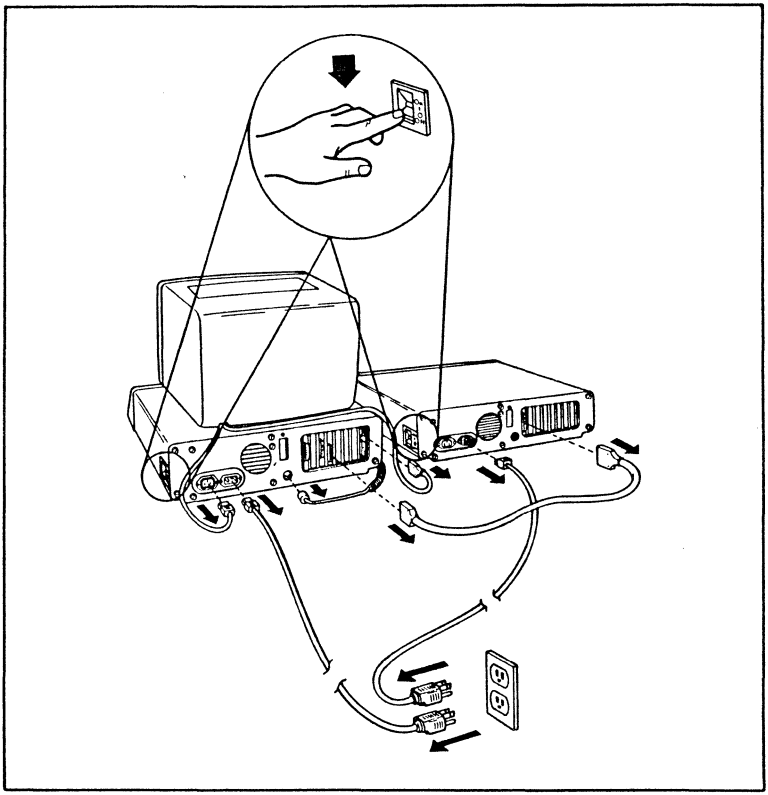


Figure 2-8. Location of Power Switches

3. Unplug the system unit power cord from the wall outlet or from the side of the system unit.
4. Position the unit to allow access to the rear.
5. Use a 1/4-in. nutdriver (or flat-blade screwdriver) and remove the five large cover-mounting screws on the rear panel of the system unit by turning the screws counter-clockwise. Save the screws for remounting the system

unit cover. See Figure 2-9.

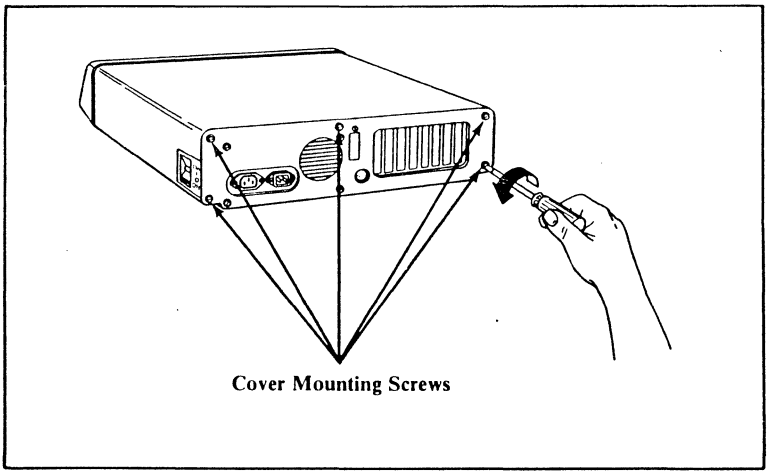


Figure 2-9. Location of Screws on Rear Panel

6. Carefully slide the unit cover away from the rear and to the front. When the cover will go no further, tilt it up, remove it from the base, and set it aside. The cover must be removed completely. See Figure 2-10.

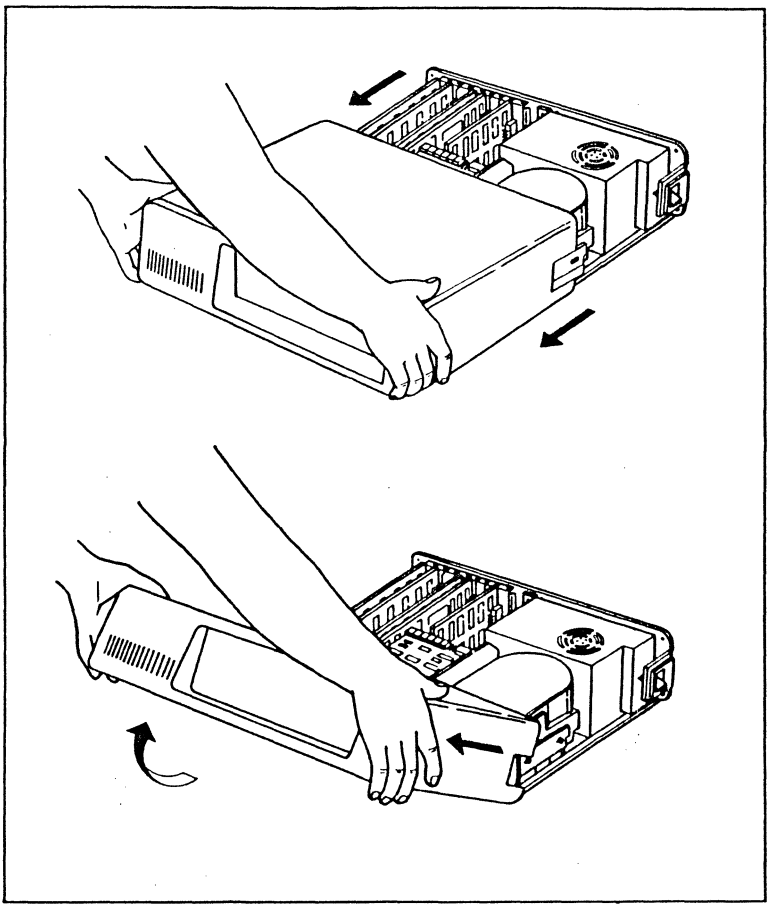


Figure 2-10. Removal of the Unit Cover

7. Look at the inside left rear of your system. You can install your Opus 100PM processor in any unused slot or pair of slots.
8. Use a 3/16-in. nutdriver (or flat-blade screwdriver) and remove the screw that holds the slot cover in place on

the inside rear panel of the system; turn the screw counter-clockwise to remove. Save the screw for installation of the Opus 100PM UNIX processor (the bracket attached to one end of the Opus 100PM processor will be used to replace the slot cover). See Figure 2-11.

NOTE: Even if the Opus 100PM requires two slots (Opus 110PM with Opus 323 memory expansion board), only one slot cover and one card guide are required. The slot cover and card guide must be attached to the main board, not the memory board.

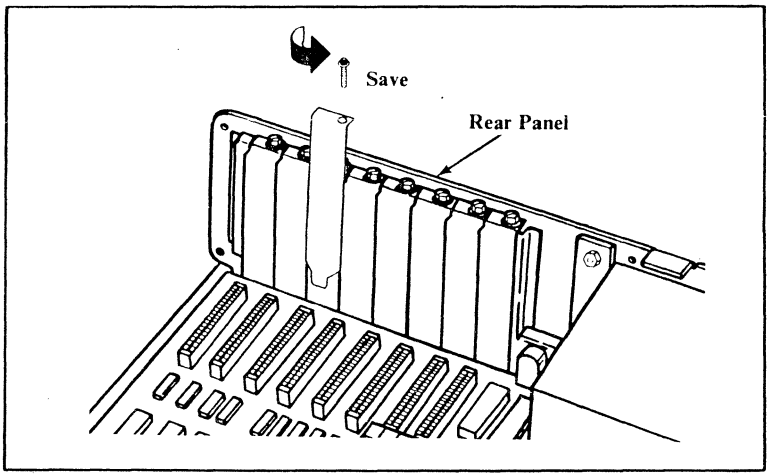


Figure 2-11. Removing Slot Cover

9. At the front of the slot, install a card guide if necessary. (This step is not necessary on the IBM PC AT.) Use the black plastic card guide supplied with the Opus 100PM processor or any compatible IBM card guide. Locate the inside front panel of the PC chassis. Notice the two rows of holes in the inside front panel — one pair for each slot. Select the pair of holes for the slot you will

use for the Opus 100PM board. Notice the guide fingers in the channel down the center of the card guide. With the fingers pointing down, install the card guide by pressing the two mounting studs on the card guide into the two holes. See Figure 2-12.

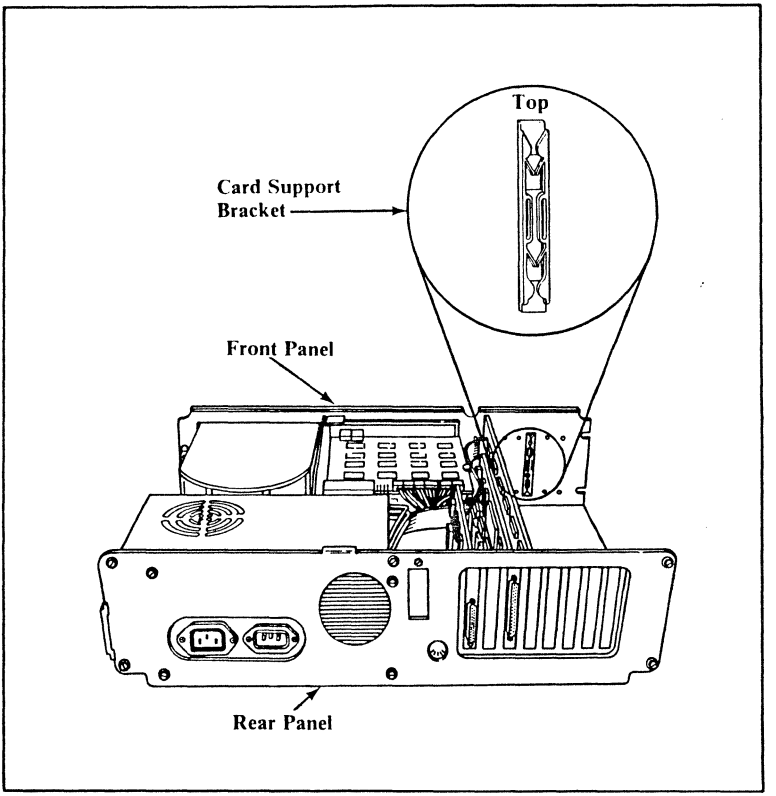


Figure 2-12. Install Card Guide

10. Hold the Opus 100PM board by the top, point the bracket end of the Opus 100PM board toward the rear of the system, and firmly press the board into the slot. If you

have a memory expansion board attached to your Opus 100PM processor, the Opus 100PM processor board (NOT the memory expansion board) should be inserted into the card guide. See Figure 2-13.

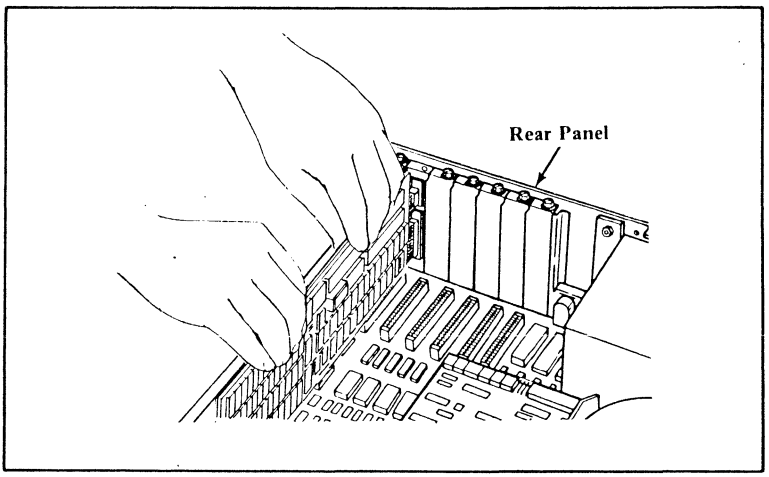


Figure 2-13. Install Opus 100PM Board

11. Align the slot in the bracket with the hole in the rear panel of the system unit. Use the screw that held the slot cover in place to attach the bracket. Tighten clockwise with a 3/16-in. nutdriver (or flat-blade screwdriver). See Figure 2-14.

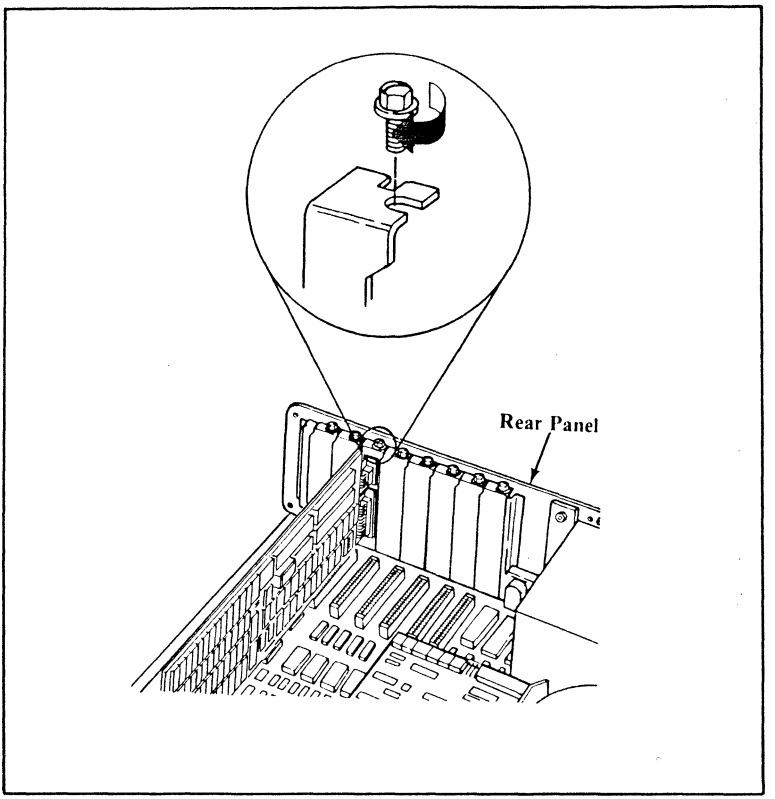


Figure 2-14. Replace Slot Cover

12. Put the unit cover on by tilting it up slightly in the front and then gradually aligning it as you slide it toward the rear of the unit. See Figure 2-15.

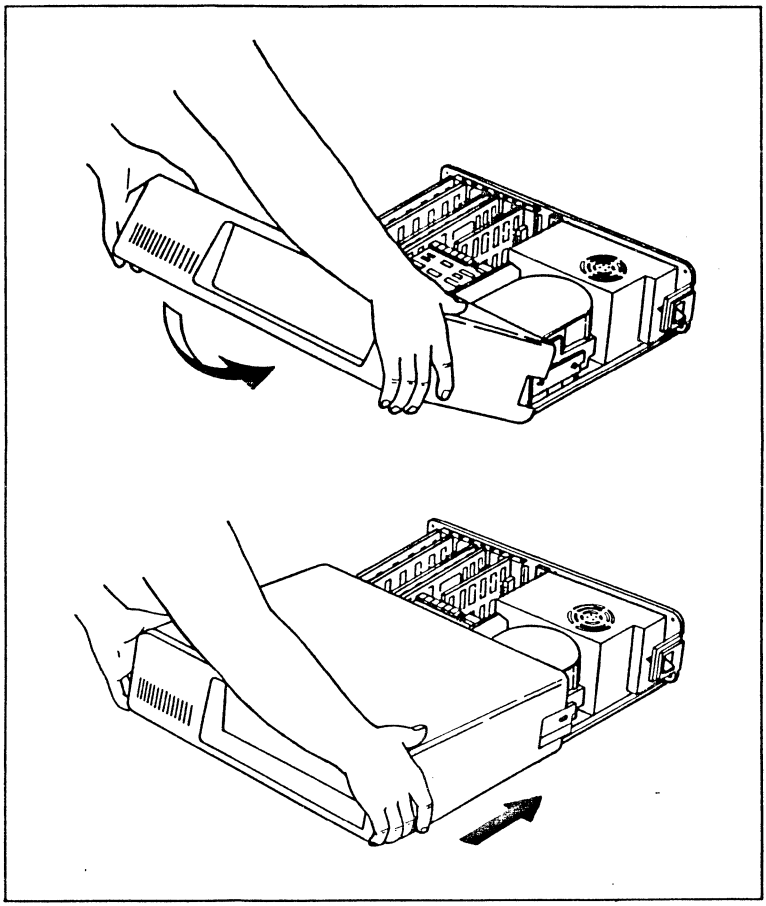


Figure 2-15. Replace Unit Cover

13. When the cover is all the way to the rear, align the five cover screws with the threaded tabs on the unit and tighten. Use a 1/4-in. nutdriver (or flat-blade screwdriver) and turn clockwise. See Figure 2-16.

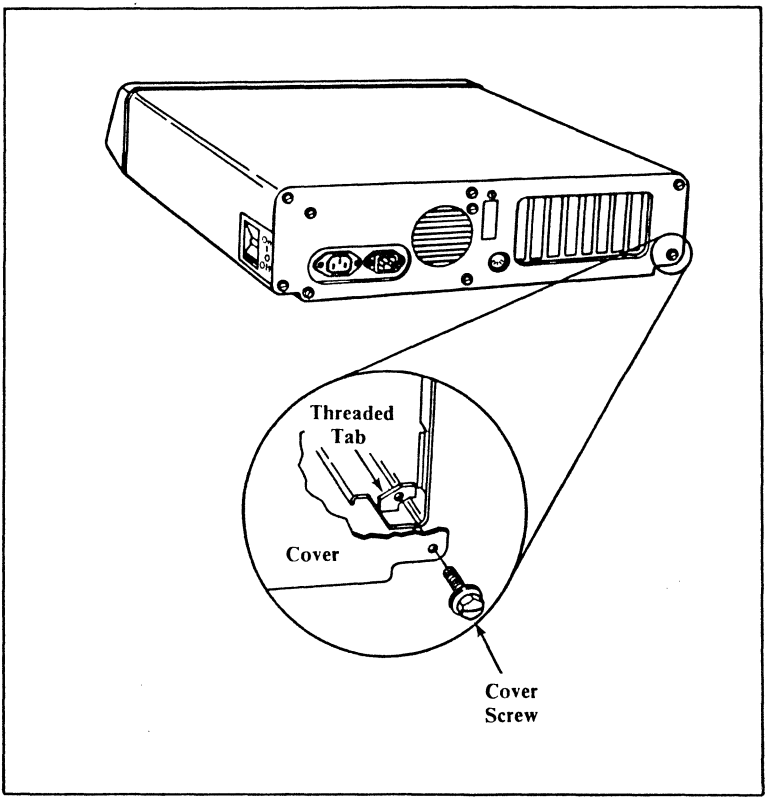


Figure 2-16. Replace Screws on Back Panel

14. Ensure that the power switch on the system unit is set to OFF.

15. Ensure that the power switch on the expansion unit (if any) is set to OFF.
16. Plug system power cord into wall outlet.
17. Set the power switch on the expansion unit (if any) to ON. If you have an expansion unit, it must be turned on before the system unit.
18. Set the power switch on the system unit to ON.

This completes the procedure for installing an Opus 100PM UNIX processor in a PC system.

3. Opus5 SOFTWARE INSTALLATION

This chapter tells how to install the Opus5 software.

Opus5 software can use the PC disk and software in either of two ways:

1. Opus5 logical disks can be separate partitions (default).
2. Opus5 logical disks can be large DOS files in the DOS partition.

Installation procedures for Opus5 differ according to whether you choose partitions or DOS files as Opus5 logical disks (guidelines follow).

If you want to use partitions for Opus5 logical disks, follow the procedures described in this chapter. The "standard installation" method, using the program **opconfig**(1*), assumes partitions. Partitions maximize performance. You MUST use partitions if you need Opus5 file systems larger than 28 Mbytes. Note that the partitions created by this procedure cannot be manipulated with normal DOS commands.

If you must use DOS files as Opus5 logical disks, follow the installation procedures described in Appendix A. We recommend that you NOT use DOS files unless you must; using DOS files can have some negative performance impact, especially for random disk I/O operations on IBM PC ATs.

You MUST use DOS files if you cannot use the standard DOS FDISK and FORMAT programs on your drive. That is, if your disk vendor provides a special FDISK and FORMAT, most likely the disk is not BIOS compatible, and you must use DOS files. See Chapter 6 for more information.

It is permissible to combine DOS-file based logical disks and partitions on the same system.

Determine now if you will use DOS files or partitions as Opus5 logical disks. If you decide to use partitions, read on. If you must use DOS files, turn to Appendix A. The rest of this chapter describes installation using partitions.

This installation procedure consists of the following steps. Use this as a checklist.

1. _____ Determine the required size of the system partitions.

2. _____ Establish the DOS partition. This involves the following steps:

   a. _____ Back up DOS (if necessary).

   b. _____ Alter the size of the DOS partition.

   c. _____ Restore DOS (if necessary).

3. _____ Issue the command **opinit install** to configure the system and begin loading system software. **Opinit**, a DOS batch file, runs the command **opconfig**(1*).

4. _____ Install Opus5 software.

In this installation procedure, the following DOS drive naming conventions are used:

Two-Drive Systems	*Four-Drive Systems*
1 diskette, 1 hard disk	2 diskette, 2 hard disks
a: = diskette drive	a: = first diskette drive
	b: = second diskette drive
c: = hard disk	c: = first hard disk
	d: = second hard disk

These naming conventions are standard for many, but not all, systems. If your system uses different names — for example, Texas Instruments systems use letters e: and f: for hard disks — be sure to use those letters instead.

If you have two floppies and one fixed disk, or two fixed disks and one floppy, use the letters for the four-drive system.

After following the installation procedures in this chapter, you should perform the administrative set-up procedures appropriate to your system. The most common of these procedures are described in Chapter 4.

Note that if you are planning to install a Priam disk at the same time as Opus software, install the Priam disk *first*; that is, follow the instructions in Section 5.10 *before* proceeding in this chapter.

3.1 Determining Required Size of System Partitions

You need at least two partitions on your system: one for DOS and one for Opus5. You probably already have a DOS partition of a certain size; if so, you might need to alter its size to make room for the Opus5 partition. This section gives guidelines for determining the respective sizes of the DOS and Opus5 partitions.

Sample configurations for systems having one standard 10 Mbyte disk (C:) are as follows:

DOS	*Opus5*	
C: 2 Mb (60 cyl)	C: 8 Mb (240 cyl)	small DOS
C: 5 Mb (150 cyl)	C: 5 Mb (150 cyl)	large DOS, very small Opus5

Sample configurations for systems having one standard 20 Mbyte disk (C:) are as follows:

DOS	*Opus5*	
C: 10 Mb (300 cyl)	C: 10 Mb (300 cyl)	large DOS
C: 2 Mb (60 cyl)	C: 18 Mb (540 cyl)	small DOS, maximum Opus5

You might need to know the desired sizes of both the DOS and Opus5 partitions for the steps that follow. You need a minimum of 5 Mbytes (10,000 512-byte blocks) to store a small Opus5 operating system. You also need an additional minimum 2-Mbyte swap area and at least 1 Mbyte for user files under Opus5. If you will be using your PC for DOS work, determine how much disk space you need for your DOS files, and compute your disk space allocation accordingly.

First determine how large your current DOS partition is, and how many free disk cylinders you have. To determine the current size of the DOS partition, type `fdisk` from DOS. Then choose menu item #4 (Display Partition Data). This shows the current disk partitioning. Find the partition whose type is "DOS". The size field in this row shows the number of disk cylinders in the DOS partition. If you subtract this size from the total disk space size, you have the number of cylinders left for the Opus partition. If you need to change the DOS partition size, go to Section 3.2. If you do not need to change the current DOS partition size, go to Section 3.3.

NOTE: Consider your choice carefully. Once the Opus5 partition has been created, its size cannot be changed without re-installing all of Opus5.

3.2 Establishing DOS Partition

NOTE: In the last section, (Determine Required Size of System Partitions), you determined the size of the Opus5 partition to be created on your disk. If there is enough room on the disk to build the required Opus5 disk partitions without overwriting the current DOS partition, you do NOT need to alter the size of your DOS partition, and you may skip to Section 3.3 (Run **opinit**).

If you are not sure how much room you have on your disk, run CHKDSK and note the figure for the total number of bytes in the partition. Subtract this figure from the total number of bytes on your disk. For example, if CHKDSK tells you your DOS partition occupies 4 Mbytes, and you have a 10-Mbyte disk, you have 6 Mbytes left for another partition.

If your DOS partition is currently not the size you want, perform the steps in this section. They are, in summary, the following:

1. Back up DOS.
2. Alter the size of the DOS partition.
3. Restore DOS.

Each is treated in a separate subsection below.

3.2.1 Backing Up DOS

CAUTION: BACKUP OF DOS FILES IS *RECOMMENDED* IN ALL CASES, BUT IT IS *REQUIRED* IF THE SIZE OF THE DOS DISK PARTITION IS TO BE CHANGED.

However, if you are installing system software from scratch (so you have no DOS files), or you have no DOS files you want to save, you can skip this step.

1. Remove unnecessary files, e.g., redundant data files, scratch files, listing files and obsolete files. In addition, files available easily from another source, such as the DOS system distribution diskettes, can optionally be removed.
2. When you are ready for backup, run **chkdsk** on the drive(s) to be backed up (see DOS manual) to determine how many diskettes you will need to hold the backup.
3. Using the DOS **format** command, format enough diskettes to hold the backup (using the size from **chkdsk**).
4. Backup the drive(s) with the DOS **backup** command (see DOS manual). For example, to backup all files on drive C: onto diskette drive A:

```
C>cd \
C>backup c: a:/s
```

This completes the procedure for backing up DOS.

3.2.2 Altering Size of DOS Partition

This section describes how to alter the size of your DOS partition.

CAUTION: THE FOLLOWING PROCEDURE WILL DESTROY INFORMATION ON THE DISK. BE SURE YOU HAVE BACKED UP ANY INFORMATION YOU WISH TO

KEEP.

In this procedure, only a DOS partition is created; the Opus5 partition(s) will be created later by **opconfig**.

If you are *increasing* the size of your DOS partition and you are deleting a non-DOS partition to make more space for DOS, be sure to delete the non-DOS partition *before* you delete the DOS partition! Use **opconfig**(1*) to delete an Opus partition.

1. Insert your DOS system diskette containing DOS, FDISK, FORMAT, RESTORE and CHKDSK into diskette drive A:.
2. Reboot DOS by typing:

   ```
   <ctrl+alt+del>
   ```

3. Use FDISK (follow instructions in DOS manual) to:
   - Delete the existing DOS partition.
   - Create a new DOS partition of the desired size.
   - Make the new DOS partition the active partition.
4. Reboot DOS by typing:

   ```
   <ctrl+alt+del>
   ```

5. Using FORMAT (see DOS manual), format the new DOS partition:

   ```
   A>format c:/s/v
   ```
   (for drive C:)

3.2.3 Restoring DOS

To restore the DOS backup files, perform the following steps:

1. Using RESTORE (see DOS manual), reload the DOS backup files:

```
A>restore a: c:/s
```
(for drive C:)

If you would like to re-install DOS, or install a new version of DOS in the new DOS partition, do so at this time. Insert your DOS diskette into drive a:, and type the following:

```
A>copy a:*.* c:
```

2. Using CHKDSK (see DOS manual), check the new DOS installation:

```
A>chkdsk c:
```
(for drive C:)

3. Remove your DOS system diskette from drive A:.
4. Reboot DOS from C: by typing:

```
<ctrl+alt+del>
```

This completes the procedure for altering the size of a DOS partition.

3.3 Running opinit

Now you are ready to begin installation of Opus5. Insert the Opus5 BOOT diskette, and type the following:

```
C>a:opinit install
```

This command does the following things:

1. It creates a directory called **\opus** under DOS on the (first) fixed disk, and copies the contents of the Opus BOOT diskette to that directory. As each file is copied, the filename is displayed. It then requests that you remove the BOOT diskette and insert the KERNEL diskette, which copies more Opus5 files to DOS, including the Opus5 kernel.

2. It runs the program **opinst**, which begins running **opconfig**(1*), the system configuration program. See the next few subsections and the manual page for in-depth information on **opconfig**.

   WARNING: **Opconfig** should be run by someone with a good understanding of the concepts of file system, and swap area, and also how **UNIX** operating systems treat devices. **Opconfig** is not a program to be run casually, since it alters the most basic system parameters and can write all over the disk(s).

3. After **opconfig** is finished, **opinit** copies a small Opus5 file system into the swap area and boots this Opus5 operating system from the swap area, bringing Opus5 to "SINGLE USER MODE". It then requests that you load the "K" diskettes; these are the minimum files required to boot Opus5 from the hard disk and run the next installation command, **/opus/bin/opload**.

3.3.1 Running opconfig for System Initialization

Opconfig sets up system parameters and configuration information. It has the following six main modules:

(1) Standard System Initialization
(2) Opus5 Device Configuration (opus.cfg)
(3) DOS/Opus5 Partition Information
(4) Disk Bad Block Handling
(5) UNIX File System Layout
(6) Opus5 UNIX System Parameters

Opinit runs **opconfig** with option (1) "Standard System Initialization" in order to begin configuration of your partition-based Opus5 system. Option (1) does the following:

- It requests your system type (e.g., TI, PC, XT, AT, etc.) and automatically adjusts device names and clock rate appropriately.
- It automatically invokes modified versions of options (2), (3), (4), and (5) as required.

The following subsections describe **opconfig** in more detail.

3.3.1.1 What Opconfig Does

Opus5 software uses information from **opconfig** during installation and booting.

Opconfig is intended to be self-explanatory for most installations. It is menu driven and has help information. See **opconfig**(1*) for more information.

Unless you have unusual system requirements, you should respond in the affirmative to all **opconfig**'s queries, making as few changes as possible to the standard definitions.

Type "exit" at any time during **opconfig** to exit the program and return to DOS.

Opconfig has a script facility that allows you to write canned installation scripts to make your customer installation even simpler. See **opconfig**(1*).

3.3.1.2 Defaults

If you answer in the affirmative consistently, you get the following by default in Opus5 Release C3:

- One partition-based Opus5 file system, which takes up the longest contiguous disk space that it can.
- A 2-Mbyte swap area, which is created within the Opus5 partition.
- A bad block area, which is 1% of the total blocks in the Opus5 file system. Sparing is done by **opconfig** as part of the standard system installation. The spare area is also within the Opus5 partition.
- The following standard system parameter settings (see **opus.cfg**(4*)):

  [TZ=PST8PDT] **Opconfig** explicitly requests you to change this if this is not correct. See **opconfig**'s help screen or Section 4.3 for format.

[i=x]	Interrupts turned off. See Section 8.2.2 and **opus.cfg**(4*).
[s=?]	Perform automatic searching for the Opus segment.
[a=128]	ALT-128 is to be the attention key sequence. (Digits must be entered on the numeric keypad.)

- In Opus5 Release C3, you get the following standard Opus5 device drivers (see Chapter 6).

```
<clock=clock>          <dos=dos>
<console=console>      <dsk/0=c:>
<flpa=a:>              <flpb=b:>
<flpA=a:bios>          <flpB=b:bios>
<tty0=com1>            <tty1=com2>
<lp=lpt1>              <lp1=lpt2>
<vdi=gssvdi>           <cgi=gsscgi>
<udio=udio>
```

Opus5 Release C2 also includes a default **opus.cfg** listing for a second hard disk (`<dsk/1=d:>`).

Note that **opconfig** explicitly requests your system type (e.g., TI, PC, XT, AT, etc.) and automatically adjusts device names and clock rate appropriately.

Note also that a disk driver name (e.g., dsk/0) corresponds to a logical disk that includes all the sections of that disk (i.e., **/dev/dsk/0s0** through **/dev/dsk/0s7**).

3.3.1.3 Running opconfig

Opconfig begins by asking your system type and adjusting your time zone if necessary. Help information is provided.

Some Opus5 system parameters are listed in the DOS file **opus.cfg**; others are part of the kernel. Part of **opconfig**'s job is to create and maintain the file **opus.cfg**; you should never have to modify that file directly in DOS (although you can).

As part of standard system initialization, **opconfig** asks you to approve, and if necessary modify, the standard mapping of kernel to **opmon** devices. See **opus.cfg**(4*) and **opconfig**(1*).

After **opus.cfg** has been set up to your satisfaction, **opconfig** reboots automatically if the configuration file has been changed. It restarts, verifies your system type, time zone, and **opus.cfg** file, and then asks if you want to use the rest of your disk for the Opus5 file system ("UNIX"). By default, one UNIX partition per physical disk is created; this is usually sufficient. If this is acceptable, you don't need to know the size of the UNIX partition, because **opconfig** will compute it automatically.

Opconfig asks the same series of questions for each logical disk specified in **opus.cfg**. See **opconfig**(1*) for more information.

After partitioning has been laid out and approved, **opconfig** inquires whether you want more than one file system per partition. One file system per logical disk is the default; but additional file systems can be created, subdividing the logical disk. You can also set your own section boundaries.

Then **opconfig** enters module (4) to spare bad blocks, and if necessary, change the size of the spare area and swap area on each partition. It first reports, for each logical disk, the percentage of blocks being used by the file system, swap area, and spare area, as shown below. Bad blocks are handled by reading the disk and creating a table at the end of the disk that contains the mapping and the alternate sectors. Some space must be allocated at the end of each partition to create this table. By default during the standard initialization procedure, 1% is allocated. This is usually sufficient for all but the worst drives.

```
The UNIX logical drive (dsk/0) is currently divided as follows:

   (1) Root file system:  16000 blocks (8 MB) 79%
   (2)        Swap area:  4000 blocks ( 2 MB) 20%
   (3) Spare block area:  202 blocks (0.4 MB) 1%

   Total partition size:  20202 blocks (10.4 MB)

Type "?" for more information, type "q" to go to the next partition,
   type the corresponding number [1-3] to change a value, or type
   "y" if these numbers are acceptable:
```

If you need more than 1% of the partition for spares, or if you want to change the size of the swap area, choose the appropriate option.

When you type "y" to signify that you are satisfied with the sizes of the spare and swap areas, **opconfig** then spares the bad blocks, if any.

Because these are potentially dangerous procedures, changing the sizes of the spare and swap areas can be done ONLY during standard system initialization.

When **opconfig** is finished, **opinit** prompts you to insert the ROOT diskette, and, eventually, the K diskettes. Opus5 is then booted in single-user mode, and you are ready to run **opload** to load the rest of the Opus5 software. Go to Section 3.4.

3.3.1.4 If Installation is Interrupted

The first part of the "opinit install" procedure copies the BOOT and KERNEL diskettes to disk. If you have interrupted the "opinit install" step after these diskettes have been loaded, and you wish to continue, you can issue the following command:

```
C>opinst
```

This restarts the installation with the **opconfig** step, allowing you to pick up the procedure without having to reload the BOOT and KERNEL diskettes.

3.4 Running opload

Execute the following command to begin loading Opus5 from the Opus-supplied diskettes:

```
# opload
```

Opload(1*) is a menu-driven, self-documenting program. The first menu lists the groups of files contained in this release of Opus5. These group names (e.g., B, E, K, R) refer to Opus distribution diskettes or groups of diskettes. Descriptions of the contents and the sizes of each group are in Appendix C. Use the **opload** menu to install files in this order:

— The required file groups B and E. (The K group is also required, but is already installed.)

— Other file groups or single files as you desire.

When you are finished, quit (q) to the Opus5 shell.

This completes the procedure for software installation of Opus5. Now perform system administration setup procedures as described in Chapter 4.

4. Opus5 ADMINISTRATIVE PROCEDURES

This chapter describes the following common Opus5 administrative procedures:

4.1 Day-to-day start-up

4.2 Multiuser mode

4.3 Setting time zone

4.4 Setting up a default login environment for root and other users (the **.profile** file)

4.5 Disabling automatic **crontab**(1) activity

4.6 Assigning passwords

4.7 Adding users

4.8 Setting terminal characteristics using **terminfo**

4.9 Setting up communications between different machines using **cu**(1) and **uucp**(1)

4.10 Adding a terminal (multiuser licensees only)

4.11 Connecting a printer

4.12 Setting up the line printer spooler

4.13 Changing the maximum file size

4.14 Backup and restore operations

4.15 Shutting down Opus5

Installations that run UNIX operating systems usually distinguish between regular users and one or more "super-users". Super-users have special access privileges; for example, some files can be edited and some programs can be run only by a super-user. Super-users generally have responsibility for system administration and maintenance. Logging in successfully with the **root** login name and password confers super-user privileges, so the super-user is often referred to simply as "root". In single-user mode (when you first bring up Opus5), you are root automatically.

4.1 Day-to-day Start-Up

Once you have installed Opus5 on your hard disk, according to the instructions in Chapter 3, the day-to-day start-up of Opus5 is simple. Only two steps are required:

1. Power up your system and start up DOS as usual. Be sure to set the time and date.
2. Type

```
C>cd \opus
C>unix
```

Messages like the following appear (messages are in **typewriter** font; comments are on the right in regular type):

```
Opus Systems

Opus5 PC Monitor 3.15          Level 1 selftest begins
Level 1 Test: PASS

Opus5 Standalone Shell 3.07    Level 2 selftest begins
Level 2 Test: PASS
Configuration: 32.32 4MB console dos dsk/0 dsk/1 flpa
            flpb lp tty0 tty1
                        System type, memory size,
                        and devices; yours may differ
```

A file system consistency check (**fsck**(1M)) is now done on the root file system (**/dev/dsk/0s0**). This is a sample display; the exact numbers for yours will be different.

```
$$  :fsck /dev/dsk/0s0

/dev/dsk/0s0
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
1051 files 13430 blocks 2966 free
```

This display illustrates a file system check with no problems. See the *UNIX System V Release 2.0 Administrator Guide* for more information on **fsck**.

Then Opus5 is booted from DOS. The system displays the amount of real and available memory in your system. This differs for different systems. A message appears announcing that you are in single-user mode, and the super-user prompt (#) appears. These displays might look like this:

```
Copyright (c) 1985, Opus Systems
Opus5 is derived from AT&T UNIX* System V

* UNIX is a trademark of AT&T Bell Laboratories

Opus5/2.0v2: Opus5C1.3
real mem = 2097152
avail mem = 1804288

INIT: SINGLE USER MODE
#
```

4.2 Multiuser Mode

We recommend you use Opus5 in multiuser mode (init state 2), even if you are the only user on the system. Multiuser mode has several important advantages:

1. The sole user possible in single-user mode is root. Multiuser mode is necessary in order for any other users — ordinary users as well as special "users" such as **uucp** and **lp** — to have access to Opus5. Running Opus5 in single-user mode can be dangerous, because root has powers and privileges not allowed ordinary users. Experienced users of **UNIX** operating systems usually prefer to run as ordinary users, in multiuser mode, to protect themselves against damaging mistakes — even if no one else is using the system.

2. The initialization file **/etc/rc** is executed when you go into init state 2. This starts up the **cron**(1) program, which is necessary to run useful, time-dependent programs such as **at**(1) and **crontab**(1).

3. You must be in multiuser mode to use the UUCP system for incoming calls. This is because UUCP requires that the user "uucp" be able to log in to Opus5. (A single-user **UNIX** license is good for two users; you need a multiuser **UNIX** license if you will have two users plus uucp.)

To start up Opus5 in multiuser mode, type **telinit 2** from single-user mode. If the date is set correctly in DOS and the time zone is set correctly in **opus.cfg**, the current date and time are substituted for "<current date and time>" in the following exchange:

```
# telinit 2
Is the date < current date and time > correct? (y or n)
```

If you type y, the system proceeds with initialization. If you type n, the system prompts:

```
Enter the correct date:
```

See the next section for how to change the time zone.

To change date or time, use the **date**(1) syntax. This changes the software date and time for DOS also.

Then the system asks:

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

If you type **y**, the system executes **fsck**(1M) on all the file systems listed in **/etc/checklist**(4) and reports results. Note that **fsck** is executed routinely when you bring up Opus5 from DOS (see 4.1 above), so if you have just brought up the system, another **fsck** is probably unnecessary unless you have more than one file system. If you type n, the system proceeds with initialization.

The next prompt is the multiuser console login prompt:

```
Console Login:
```

The login message for ordinary terminals is "`login:` ". This message appears on all other terminals at the same time logins are enabled for the console.

To make Opus5 come up in multiuser mode by default, modify the file **/etc/inittab**(4), changing the first line from

```
is:s:initdefault
```

to

```
is:2:initdefault
```

See the document "Single User and Multiuser" in the *UNIX System V Release 2.0 Operator Guide*, **init**(1M), and **inittab**(4) for further background information on the **init** states.

To make Opus5 accessible to ordinary users, you must create logins for them (see Section 4.7). If security is important for your site, you should also give "root" a password, and insist that ordinary users have passwords too. Section 4.6 describes how to assign passwords.

4.3 Setting Time Zone

Opus5 keeps time internally in GMT (Greenwich Mean Time); it is up to individual Opus5 sites to tell Opus5 what time zone they are really in. The Opus5 default time zone is Pacific Time. If you are not in Pacific Time, you should edit several files and change the time zone specified there. The files that should be changed are:

/etc/bcheckrc

/etc/rc
/etc/profile
$HOME/.profile

Use **vi**(1) or **ed**(1) to edit Opus5 files. Instructions are given below. $HOME means the home directory of each user.

The DOS file **opus.cfg** sets the time zone for **opmon**, but most probably you do not need to edit that file now, because the file was edited automatically by **opconfig** during system installation. If you did not respond to **opconfig**'s query regarding your time zone, or you want to change your response, you should re-run **opconfig** to make this change to **opus.cfg**. **Opconfig** asks for time zone as part of module 1, "Standard System Initialization".

Note that nothing disastrous happens if you don't change the time zone specifications listed above; Opus5 and DOS still run. Problems might arise because Opus5 usually puts a time-stamp on files and output. **Make**(1) depends on the date. Also, **crontab**(1) and **at**(1) use the Opus5 date and time to run programs automatically; if the date and time are incorrect, these programs might run programs automatically at the wrong times.

A sample Opus5 time zone specification is the following:

```
TZ=PST8PDT
```

The format is *TZ=xxxnzzz* where "xxx" is the standard local time zone abbreviation, "n" is the difference in hours from GMT, and "zzz" is the abbreviation for the daylight-saving local time zone, if any. (See **date**(1).) "n" can be two digits.

The TZ entry shown above (PST8PDT) is for Pacific Standard Time, 8 hours west of Greenwich Mean Time, with Daylight Savings Time.

In the continental US, the entries are:

[TZ=PST8PDT]	[TZ=PST8]
[TZ=MST7MDT]	[TZ=MST7]
[TZ=CST6CDT]	[TZ=CST6]
[TZ=EST5EDT]	[TZ=EST5]

Use the form on the right in areas that do not observe daylight savings time, or where time changes are non-standard. For example, if you are in Pacific time zone but your area observes non-standard changeover dates, use:

```
[TZ=PST8]
```

during standard time and

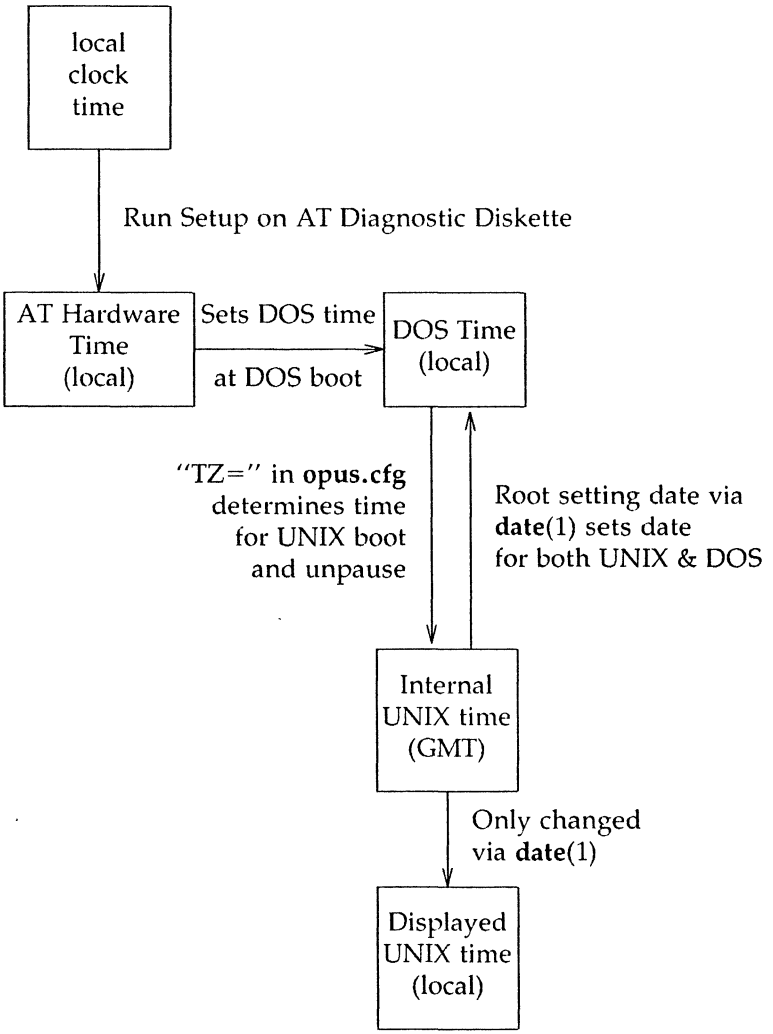
```
[TZ=PST7]
```

during daylight time, in all relevant files.

The number portion is negative in areas east of GMT and west of the international date line.

4.3.1 Daylight Savings Time

The following diagram gives the complete picture of how DOS and UNIX interact on an AT or clone to handle the time and date.



Transitions to and from Daylight Savings Time are not handled transparently in the Opus 100PM system. This is because the AT hardware clock is the ultimate keeper of time on the AT; and you can set the AT hardware clock ONLY by

running SETUP from the AT diagnostic diskette.

When DOS boots, it gets its time from the AT hardware clock. From DOS, you can also change the time using the DOS DATE and TIME commands. And the DOS time is also changed if you issue a UNIX **date**(1) command. So from within DOS, you can alter the time DOS receives from the hardware clock. But you can't alter the hardware clock time except using the AT SETUP program.

UNIX's internal time is Greenwich Mean Time (GMT). When UNIX boots, it uses the TZ variable in **opus.cfg** to convert the current DOS time to GMT for its own internal use. It displays the local time by doing the appropriate calculations on the GMT. The UNIX time (GMT) is set again after resuming from a pause to DOS. (By "pause", we mean typing some variant of the **dos**(1*) command, and resuming UNIX by typing "**resume**" or "**exit**" or "**unix**". The time is not re-set after "**DOS BG**", because there's no need to; UNIX hasn't stopped receiving clock ticks.) The UNIX time is also set by the UNIX **date**(1) command. In other words, the UNIX time always starts with the current DOS time; from UNIX, it's impossible to change the AT hardware clock.

Thus, you can't change the AT hardware clock either from DOS or UNIX.

This is why even though UNIX *does* automatically change its time at Daylight Savings transitions, this is only a temporary solution. The next time you boot DOS, the DOS time changes back to the AT hardware time. And then, unless you manually change the time in DOS, UNIX will start up with the wrong time.

The only "permanent" solution for Daylight Savings Time transitions is to change the AT hardware time using SETUP. And this solution only lasts until the next transition.

4.4 The .profile File

The **.profile** file in each user's home directory initializes the login environment for each user. Standard **UNIX** System V

has certain default settings that generally are customized for each site and each user. The following is the Opus-supplied **.profile** entry supplied as both **/opus/.profile** and **/.profile** (i.e., this is the default .profile for root and any other user whose home directory is /). Each line is explained below.

```
stty kill '^x' erase '^h' echoe
  PATH=.:/opus/bin:/bin:/usr/bin:/etc; export PATH
  TZ=PST8PDT; export TZ
  TERM=opus-pc; export TERM
  EXINIT='set redraw nows sm'; export EXINIT
  dir() ls -CF $@;
```

stty — Sets terminal characteristics. This **stty**(1) entry does the following:

- sets the line-cancel character to ^X,
- sets the backspace character to ^H (PC backspace key), and specifies that the backspace function erase the characters backspaced over.

The default **stty** settings work with standard IBM PC terminals. Other settings may be appropriate, depending on your terminal. See Section 4.10.

In a multi-user environment, one Opus system might service many users, some with different kinds of terminals. Problems might arise if a user whose **.profile** is appropriate for terminal *X* logs into terminal *Y*. See Section 4.4.1 for a handy way to ensure that the **stty** settings remain correct, even in this case. The method described in Section 4.4.1 associates **stty** settings with the actual terminal type, instead of the user's login profile.

PATH — Sets the directories that are to be searched for commands that you issue. This entry for PATH starts searching in the current directory; note that this is not standard for **UNIX** operating systems, although it is standard for DOS, regardless of the PATH specified.

TZ — Sets the local time zone for conversion of Opus5 internal GMT time to local time. See Section 4.3 above.

TERM — Defines the characteristics of this user's terminal for programs such as **vi**(1) and **pg**(1) that use the **terminfo**(4) terminal interface.

The entries `opus-pc` and `opus-ansi` are Opus-supplied terminal definitions for the console terminal as supported by Opus5. `Opus-pc` is used for `<console>` and `opus-ansi` is used for `<doscon>`. See Chapter 6 for more information on devices. See Section 4.8 for more information on **terminfo**.

EXINIT — Defines the environment for **ex**(1) and **vi**(1). See **ex**(1).

dir — This entry defines a shell function for listing directories. See **sh**(1) and **ls**(1).

4.4.1 Multiple Users with Assorted Terminals

Suppose you have an 8-MHz AT with an Opus 110PM processor and 24 serial ports. Many types of terminals can be used with this machine; suppose your company has several different types, but most of your users are not UNIX experts. You don't want them to have to worry about setting up their `.profiles` correctly, especially with regard to terminal types and **stty** settings. Here's a solution.

First create a file called **/etc/lines**, which lists all the TTY ports, their types, and usual users. (The user names are in the file only so you can remember which cable goes to which desk.) The following is a sample **/etc/lines** file:

```
/dev/console:opus-pc:

/dev/tty0:wy30:glennb:
/dev/tty1:wy30:debbie:
/dev/tty2:wy60:glenn:
/dev/tty3:opus-pc-ic:craig:
/dev/tty4:wy60:tom:
/dev/tty5:vt100:rick: # Cross Talk
/dev/tty6:wy60:bill:
/dev/tty7:wy30:joan:
```

Then create one central `.profile`, in the directory `/opus`. Each user's personal `.profile` executes this central `.profile` (contains the line `. /opus/.profile`).

In `/opus/.profile`, listed below, assign to the variable TTY the result of executing the command **tty**(1). This puts the name of each user's TTY device into the variable TTY.

```
PATH=.:$HOME/bin:/opus/bin:/bin:/usr/bin; export PATH
TZ=PST8PDT; export TZ
TTY=`tty`; export TTY
TERM=`grep "^$TTY:" /etc/lines | cut -f2 -d:`; export TEI
stty erase ^h kill ^x echoe
case "$TERM" in
       opus-pc)
              stty tab0;;
       opus-pc-ic)
              stty tab0 -parenb cs8 -istrip;;
       wy60)
              echo "\033c8"`date "+%H%M"`"\c"
              echo "\033\`c\c";;
esac
EXINIT='set redraw nows sm ai'; export EXINIT
dir() /bin/ls -CF $@;
```

Then you can derive the variable TERM by running **grep**(1) on the **/etc/lines** file for TTY (the user's device name), and **cut**(1) out the terminal type field. This puts the terminal type (`vt100`, `wy60`, `wy30`, etc.) into the variable TERM. This is exactly what you need. Now just include a `case` statement that performs the necessary setup for each terminal type. Controlling the terminal setup via TTY port number allows any user logging in to any terminal to get the right terminal information.

4.5 Disabling Automatic Crontab Activity

UNIX is designed as a time-shared operating system, and standard Opus5 software reflects this. Standard Opus5 contains **crontab**s(1) which automatically perform functions useful in a time-sharing environment, such as accounting and

system activity reporting. Symptoms of automatic **crontab**s include "spontaneous" disk activity, especially on the hour, and lots of mail for root, adm, and sys. These features consume CPU cycles and disk space, and might be only marginally useful in a single-user Opus5 workstation. Therefore, you might want to disable the **crontab**s that carry out these automatic functions, especially if you do not install the accounting diskette.

These **crontab**s can be disabled in a number of ways.

1. The most straightforward method is to log in as super-user and remove the files **adm**, **root**, **sys**, and **uucp**, in the directory **/usr/spool/cron/crontabs**. We recommend this procedure if you are certain you do not want these files. (Even if you change your mind later, you can always get them back from your Opus5 release diskettes using **/opus/bin/opload**.)

   Beware of issuing the "rm *" command in the **crontabs** directory if users have set up personal **crontabs** there.

2. **Crontab** filenames must be valid user names; so **crontab**s don't work if the names are changed to invalid user names. Thus, another way to disable **crontab** activity is to change the filenames in the directory **/usr/spool/cron/crontabs**. For example, you can do the following:

   ```
   # cd /usr/spool/cron/crontabs
   # mv root root.stock
   # mv adm adm.stock
   # mv sys sys.stock
   # mv uucp uucp.stock
   ```

3. **Crontab**s are ignored if they reside in directories other than **/usr/spool/cron/crontabs**. Thus, another way to disable automatic **crontab** activity is to move the offending files out of **/usr/spool/cron/crontabs**.

   The second and third methods are recommended if you are not sure you want to remove the system default **crontab**s.

4.6 Assigning Passwords

If root has no password, anybody can successfully log in as root. This is dangerous, since root has privileges and powers not available to any other user. If inexperienced or malicious users pose a problem for your site, you should give root a password, keep it secret, and change it regularly.

Root assigns a password to him/herself in the same way any other user does — with the **passwd**(1) command. Login as root and type

```
# passwd
```

The system responds

```
Changing password for root
New password:
```

Type the password. The **passwd** program asks you to type the password a second time for verification purposes.

Other users can change password by following the same procedure, except the user logs in as him/herself, types **passwd**, and the system responds "`Changing password for <user name>`", where <user name> is the user's login name. Ordinary users' passwords must be more than six characters long, and must contain at least one non-alphabetic character.

Root can change anybody's password; ordinary users can change only their own.

4.7 Adding Users

You can add users to your system by following the four steps summarized here.

1. Put a line in **/etc/passwd** for the new user (see 4.7.1).
2. Create a home directory for the new user (see 4.7.2).
3. Give the new user a **.profile** (see 4.7.3).
4. Make the new user the owner of his/her home directory (see 4.7.4).

Each step must be performed by root, and is described in detail in separate subsections that follow.

We recommend that you install yourself as an ordinary user of your system and do most of your work as an ordinary user, even if you are the only user. This can prevent you from accidentally damaging your system.

In addition to login permissions and super-user permissions, some Opus5 facilities such as **uucp**(1), **at**(1), and **crontab**(1) have a separate permission structure; users must be granted permissions to use these facilities independently of other permissions they might have. See the AT&T documentation on these facilities for further information.

4.7.1 Adding a Line to /etc/passwd

All legitimate users of your Opus5 system must be registered in the file **/etc/passwd**. This file is writeable only by root. This is a sample **/etc/passwd** file:

```
root:Xdfhjak47891:0:3:0000-Admin(0000):/:
daemon:hj784sdj!$%:1:12:0000-Admin(0000):/:
bin:GH*&D234dg1:2:2:0000-Admin(0000):/bin:
sys:jf8ghd6e&^b:3:3:0000-Admin(0000):/usr/src:
adm:aw3%#yh9jk:4:4:0000-Admin(0000):/usr/adm:
uucp:xmYUdhd56$^:5:1:0000-uucp(0000):/usr/lib/uucp:
nuucp::6:1::/usr/spool/uucppublic:/usr/lib/uucp/uucico
sync::20:1:0000-Admin(0000):/:/bin/sync
rje:pdt27(^g0:68:8:0000-rje(0000):/usr/rje:
shqer:"!hsjhs85a4:69:8:0000-rje(0000):/usr/rje:
lp:b7&fwao)#:71:2:0000-lp(0000):/usr/spool/lp:
lename:bsuwe530-12:110:6::/user/lename:
tester:5DRos79hchg:111:5::/user/tester:
```

Each line has seven fields separated by colons. The fields are:

1. login name
2. encrypted password
3. user id (numbers 0-99 reserved)

4. group id
5. accounting information
6. login directory
7. program name to be executed when this user logs in

Fields 2, 5, and 7 are optional, i.e., a user can login successfully even if they are missing. Field 7 defaults to **/bin/sh**.

This sample shows a **passwd** file that is just like the standard released one, except that every user has been given a password, and two users (lename and tester) have been added.

If you are concerned about security at your site, you should assign passwords to all users, even those user names supplied by the system (like **sys, bin, daemon,** etc.).

Group ids are used by systems that maintain groups of users. The permission scheme of Opus5 supports the assignment of permissions based on *group* membership. Groups are listed in the file **/etc/group**.

See the *UNIX System V Release 2.0 Administrator Guide* document on "Setting Up the UNIX System", and the manual pages **passwd**(1), **login**(1), **passwd**(4), and **group**(4) for additional information.

4.7.2 Creating a Home Directory

Determine where you want the new user's home directory, and use **mkdir** to create a directory there. You must be logged in as root for this procedure. For example, if you want to place the user lename's directory under **/user**, type

```
# mkdir /user
# cd /user
# mkdir lename
```

Users' home directories typically have the same names as their login names.

4.7.3 Giving User a .profile

The file **/opus/.profile** (see Section 4.4 above) provides a serviceable **.profile** for many sites. You can also write one tailored to your liking. To give a user a **.profile**, you merely copy a **.profile** to his/her home directory, and, if necessary, modify it appropriately.

For example, to give lename the /opus/.profile, when his home directory is /user/lename, issue the command:

```
# cp /opus/.profile /user/lename/.profile
```

4.7.4 Changing Ownership of User's Home Directory

New users should own their home directories, so they can set permissions on them as they like. To give the user lename ownership of the directory /user/lename, issue the command:

```
# chown lename /user/lename
```

This completes the procedure for adding a user.

4.8 Using terminfo

The files in **/usr/lib/terminfo/*/*** describe terminals. Certain programs such as **vi**(1) and **pg**(1) use these terminal descriptions. **Terminfo** source entries are coded in a special syntax, examples of which are given below; these entries list the capabilities of a terminal and describe how the terminal performs certain operations. See **terminfo**(4) and **tic**(1M) for more information and examples.

Opus supplies **terminfo** source entries for the PC console. The entry *opus-pc* is used with the **opmon** <console> (BIOS) driver; the entry *opus-ansi* is used with the **opmon** <doscon> (DOS) driver. For *opus-ansi* to function, the DOS driver *ansi-sys* must be specified in the DOS file *config.sys*.

The **terminfo** entries are as follows:

```
opus-pc,
   cr=^M, ind=^J, bel=^G, cols#80, lines#25, nel=^M^J,
   cub1=^H, cuf1=\E[C, cuu1=\E[A, cud1=^J, ri=\EM,
```

```
    cub=\E[%p1%dD, cuf=\E[%p1%dC, cuu=\E[%p1%dA,
    cud=\E[%p1%dB, cup=\E[%i%p1%d;%p2%dH,
    il1=\E[L, dl1=\E[M,
    home=\E[H, clear=\E[2J, el=\E[K, ed=\E[J, ll=\E[25H,
    smso=\E[1m, rmso=\E[0m, smul=\E[4m, rmul=\E[0m,
    bold=\E[1m, rev=\E[7m, blink=\E[5m, sgr0=\E[0m,
    kbs=^H, am, vt#3,
    sc=\E[s, rc=\E[u,
    km, kdch1=\037\123, kich1=\037\122,
    kcuu1=\037\110, kcub1=\037\113, kcuf1=\037\115,
    kcud1=\037\120,
    khome=\037\107, knp=\037\121, kpp=\037\111,
    kf1=\037\073, kf2=\037\074, kf3=\037\075, kf4=\037\076,
    kf5=\037\077, kf6=\037\100, kf7=\037\101, kf8=\037\102,
    kf9=\037\103, kf10=\037\104,
opus-ansi,
    cr=^M, ind=^J, bel=^G, cols#80, lines#25, nel=^M^J,
    cub1=^H, cuf1=\E[C, cuu1=\E[A, cud1=^J, ri=\EM,
    cub=\E[%p1%dD, cuf=\E[%p1%dC, cuu=\E[%p1%dA,
    cud=\E[%p1%dB, cup=\E[%i%p1%d;%p2%dH,
    home=\E[H, clear=\E[2J, el=\E[K, ed=\E[J, ll=\E[25H,
    smso=\E[1m, rmso=\E[0m, smul=\E[4m, rmul=\E[0m,
    bold=\E[1m, rev=\E[7m, blink=\E[5m, sgr0=\E[0m,
    kbs=^H, am, vt#3,
    sc=\E[s, rc=\E[u,
    km, kdch1=\037\123, kich1=\037\122,
    kcuu1=\037\110, kcub1=\037\113, kcuf1=\037\115,
    kcud1=\037\120,
    khome=\037\107, knp=\037\121, kpp=\037\111,
    kf1=\037\073, kf2=\037\074, kf3=\037\075, kf4=\037\076,
    kf5=\037\077, kf6=\037\100, kf7=\037\101, kf8=\037\102,
    kf9=\037\103, kf10=\037\104,
```

These sources are in the file **/opus/src/opus.ti**.

If you will use a terminal other than the standard PC console, search the directories under **/usr/lib/terminfo** for a **terminfo** entry that applies to your terminal. If no match is found, you might have to write your own **terminfo** description. See **terminfo**(4). Compile the source with **tic**(1M), and place a reference to your new terminal description in your **.profile**.

4.9 Setting Up UUCP

This section summarizes the steps required to set up a UUCP link between your computer and another computer running a **UNIX** operating system. As an example, we use a relatively simple case: connecting an IBM PC XT to a Hayes 1200 Smartmodem for both dialin and dialout UUCP connections.

A multiuser **UNIX** license is required if more than two users will log in to your system. Another computer logging in via a UUCP link counts as one user.

4.9.1 Hardware Considerations

You need a modem and a cable to connect your PC to the modem. If you have a Hayes or Hayes-compatible modem, you can follow the examples in this section and use a simple extension cable, i.e., no wires switched. If you are using a different modem, check your vendor documentation for cabling requirements.

The modem should do the following:

- It should control the hang-up function with DTR (Data Terminal Ready); that is, it should hang up when DTR is off, and take the phone off-hook when DTR is on.
- It should tell the computer if a Data Carrier is detected via DCD (Data Carrier Detect).

The Hayes Smartmodem 1200 requires the following configuration switches set. Note that different Hayes modems set switches differently; on some, the up position means ON, while on others it means OFF. Check your modem to be sure.

Position	*Setting*	*Meaning*
1	ON	Computer supports DTR
2	*	Send results as English words
3	ON	Do not send result codes
4	OFF	Do not echo characters
5	**	Answer incoming calls
6	ON	Report DCD
7	**	RJ11 telephone jack
8	OFF	Enable modem command recognition

* Setting does not matter if switch 3 is ON.
** Site-dependent

If you are using a different kind of modem, set the switches for the same functions according to the vendor documentation. If your modem is unable to supply DCD, connect the DTR line back to the DCD line at the computer, to simulate constant DCD, or use **tty128** instead of **tty0**, or **tty129** instead of **tty1**, etc. (see Section 6.7).

4.9.2 Software Checklist

This section summarizes the steps required to set up UUCP from the software side. For further information, see the documents "UUCP Administration" in the *UNIX System V Release 2.0 Administrator Guide*, and "UNIX System to UNIX System Copy (UUCP)" in the *UNIX System V Release 2.0 Support Tools Guide*. See also **dial**(3C*).

1. Verify your system nodename using **uname**(1) with the "-n" option. If you don't have a system nodename, assign one using **opconfig**.
2. Verify that the **/etc/passwd** entry for the user nuucp has **/usr/lib/uucp/uucico** as its shell (the 7th field).
3. Define a login password for uucp and nuucp using the **passwd** command (see Section 4.6).
4. Modify **/etc/inittab** to set the baud rate for your dialin line. For example, if your modem uses 1200 baud, the

getty specified for the line for that port, say tty0, should specify 1200 baud, as in the following example:

```
00:2:respawn:/etc/getty -t20 tty0 1200
```

For more information on **inittab**, see the document "Setting Up the UNIX System" in the *UNIX System V Release 2.0 Administrator Guide*, and the manual pages **getty**(1M) and **inittab**(4).

5. Using the command "`ls -l`", verify that all files in the directory **/usr/lib/uucp** are owned by uucp. Correct with **chown**(1) if necessary.

6. Modify **/usr/lib/uucp/USERFILE** to record information about sites that will call you. If you will be called by the system name *pip*, logging in to your system as *dip*, permitted to transfer files to and from **/usr/spool**, the entry looks like this:

   ```
   dip,pip /usr/spool
   ```

   USERFILE should have mode 400 (readable only by root).

7. Modify **/usr/lib/uucp/L.sys** to record information about sites that you will call. The format of lines in **L.sys** is the following:

   ```
   nodename time device class phone_# login_info
   ```

   where

   nodename — the nodename of the system you are calling.

   time — times acceptable for calling. "Any" here means any time is all right. Days are abbreviated

   ```
   Su Mo Tu We Th Fr Sa
   ```

   or "Wk" for any weekday. The time should be given as a range of times (e.g., 0800-1230). If no time is specified, "Any" is assumed.

device — either "ACU" if you are using one; or the name of the port to which your dial-out modem is connected (e.g., tty128); or a hardwired device name, if the connection is direct.

class — the line speed, usually 300 or 1200 baud for phone connections.

phone_# — the phone number of the modem at the site you are calling. For hardwired devices, this field contains the same string used in the "device" field.

login_info — a series of "expect-send" fields, where "expect" is what you expect to read when you connect with the remote site (e.g., "login:"); and "send" is what you will send when the "expect" string is received (e.g., your userid). Passwords are also normally expected.

A sample **L.sys** line to call out using an ordinary TTY dial-out port (say, tty128) might be the following:

```
pip Any tty128 1200 555-1212 ogin-EOT-ogin yrid ssword yrpw
```

This says that you contact the site whose nodename is "pip". Their phone number is 555-1212, and they have given you the userid "yrid" and the password "yrpw".

Like **USERFILE**, **L.sys** should have mode 400.

8. Modify the file **/usr/lib/uucp/L-devices** to record information about your ACU or direct line. The format of lines in **L-devices** is the following:

```
type line call_unit speed
```

where

type — either ACU or DIR. Use "ACU" if you are calling out with an ordinary dial-out modem. (This assumes you have written a

dial program according to **dial**(3C*)).

line — the device you are using, e.g., tty128.

call_unit — the name of your dial program or "0" for direct lines.

speed — the line speed.

For example, if you use the dial program **/opus/bin/hayes** and tty128 at 1200 baud, your **L-devices** line would look like the following:

```
ACU tty128 /opus/bin/hayes 1200
```

A **termcap** is a terminal description file (like **terminfo** files in System V) used in System III and UCB versions of **UNIX** operating systems. The file **/opus/src/opus.tc** contains a **termcap** for standard PC consoles. If you call sites that use **termcap**, you will need this file to set up terminal characteristics to enable remote **vi**(1).

4.10 Adding a Terminal

This section outlines the steps required to add a terminal to your Opus 100PM system. Much of the discussion here is relevant to adding a serial printer also; see the next section for a summary of the steps required to connect a serial printer.

The system configuration file **opus.cfg** defines two serial lines by default: DOS `<com1>` corresponds to **/dev/tty0**, and DOS `<com2>` corresponds to **/dev/tty1**. More serial lines can be added if you install an adapter board, and make appropriate modifications to **opus.cfg** using the system configuration program **opconfig** (see Section 6.6 and Appendix E).

To add a terminal in addition to the console (multiuser license is required), perform the following steps:

1. Connect the terminal to one of the PC serial ports. A null modem is required, because the standard serial ports on the PC are designed to be connected to communications devices (e.g., modems). A null modem is a device that reverses the pin configuration so that the port can be

connected to devices other than communications devices.

2. Read **gettydefs**(4) and examine the file **/etc/gettydefs**. This file determines the terminal characteristics for terminals of various speeds.

   Most probably, the line in **gettydefs** that corresponds to the speed of your terminal will work without modification. Scan the file briefly at this time to see if you can spot any obvious problems.

3. Add a line to the file **/etc/inittab** to enable logins on the new terminal. The line should have the following form:

```
<id>:<rstate>:respawn:/etc/getty -t<timeout> tty<n> <speed>
```

where

id	one or two characters that uniquely identify this **inittab** entry.
rstate	the run level in which this entry is to be processed. Terminals usually come up in init state 2 (multiuser), so rstate should be 2.
timeout	an option to getty; tells how long in seconds you want getty to wait before exiting if the line is opened (the login prompt appears) but no one types anything.
n	the number of this TTY in **/dev**, e.g., for **/dev/tty1**, n=1.
speed	the baud rate for this line. (This is actually the first word of the appropriate **gettydefs** line for this terminal.)

For example, the entry for **/dev/tty0**, timeout=60 seconds, line speed=9600 baud, might look like this:

```
00:2:respawn:/etc/getty -t60 tty0 9600
```

See **inittab**(4) for further information.

4. Issue the command **telinit q** to cause the system to read the modified **inittab**. The login prompt should appear on the new terminal.

   If you have trouble logging in once the login prompt appears, go back and read the file **/etc/gettydefs** and do one of the following:

   - If you can, adjust your terminal according to the line in **gettydefs** that corresponds to the speed you intend to use. Consult the vendor documentation for your terminal for instructions on how to adjust your terminal.
   - OR, if your terminal cannot be changed to correspond to the appropriate line in **gettydefs**, modify **gettydefs**, and try "telinit q" again. See Section 4.10.1 for advice on how to modify **gettydefs**.

5. The TERM definition in the **.profile** file for users of the new terminal must match the new terminal.

6. If the terminal behaves strangely once you have logged in, you can change its characteristics, either for your current Opus5 session only, or for any Opus5 session, using either **stty**(1) or some combination of **gettydefs** and **stty**. See Section 4.10.1.

   You can also use **stty** to establish settings for non-login serial ports (such as printer ports). See Sections 4.10.1 and 4.11 for more information.

4.10.1 Modifying /etc/gettydefs and Using stty

Most likely, you will NOT need either to modify **/etc/gettydefs** or use **stty**. You need to consult this section only if modifications are required.

The flags set in the **termio**(7) structure in the TTY driver determine the current terminal characteristics. The programs **getty** and **stty** both set flags in the **termio** structure.

Getty, using the file **/etc/gettydefs**, sets terminal characteristics for login (according to **gettydefs**'s "initial

flags"); and sets initial terminal characteristics for interaction with the shell (according to **gettydefs**'s "final flags"). **Getty** modifies **termio** "from scratch"; that is, **termio** has no flags set (contains all zeroes) when **getty** begins.

Once login is complete, you can use **stty** to modify terminal characteristics. The **termio** that **stty** receives to work on contains the "final flags" settings from **gettydefs** and zeroes for all other flags.

Thus, **getty** and **stty** are somewhat redundant in function. Both modify the **termio** structure, and thus can change the current terminal characteristics. One difference between them arises from the fact that the **termio** structure is always re-initialized when a terminal is closed. Thus, both **getty** and **stty** settings are thrown away when you log off; but **gettydefs** settings are kept in the file **/etc/gettydefs**, and are used again by **getty** every time it opens the port. Thus, if you need terminal characteristics to be the same for every session, you might want to specify **gettydefs**'s "final flags" carefully, in order to avoid having to tweak settings with **stty** later. (This issue can also be handled in other ways, e.g., by issuing **stty** commands in /**etc/rc** or **.profile** files.)

Another difference between **getty** and **stty** is that some **termio** fields can be set only with **stty**, in particular, the erase and kill characters.

Another important difference exists in syntax. You can turn off an **stty** option by putting a minus in front of it. For example,

```
stty -parenb < /dev/tty0
```

disables parity. However, you cannot use the minus convention in **gettydefs**. In **gettydefs**, "-PARENB" has no effect.

Both **getty** and **stty** can be invoked with the SANE option, which establishes the following terminal characteristics:

```
ISANE:  BRKINT  IGNPAR  ISTRIP  ICRNL  IXON
OSANE:  OPOST  ONLCR
CSANE:  CS7  PARENB  CREAD
LSANE:  ISIG  ICANON  ECHO  ECHOK
```

Note that by default, **gettydefs** always specifies SANE in the "final flags". These characteristics are adequate for most ASCII terminals.

If you can't modify your terminal to match a **gettydefs** line, you must modify **/etc/gettydefs** to match your new terminal. We can provide only general guidelines for this, since different terminals require different settings. You must use the vendor documentation supplied with your terminal to determine how to modify it. The basic idea is simply to be sure the terminal settings and the **gettydefs** settings match. This might require some experimentation. The manual entry **termio**(7) describes the various possible settings and what they do.

If SANE does not meet your needs for "final flags", you must either change the final flags in **gettydefs**, or issue **stty** commands after login (e.g., in your **.profile** file). We present an example here of changing the final flags in **gettydefs**; suppose that (for some reason) you want to disable flow control, i.e., disable cntl-S/cntl-Q for stopping and starting output. Flow control is governed by IXON and IXANY; this means you want everything else in SANE except the IXON option. This is only an example to illustrate the common mistakes that are made in modifying **gettydefs**.

You might think you could just turn off IXON and IXANY with a **gettydefs** line such as the following:

```
1200# B1200 CS7 # B1200 SANE -IXON -IXANY ...
                                    WRONG!
```

As mentioned above, the "-" convention, which works for **stty**, does nothing in **gettydefs**.

To get rid of IXON and IXANY plus all the other SANE options, you must specify all the SANE options individually (leaving out IXON and removing IXANY) as follows.

```
1200# B1200 PARENB HUPCL CS8 # B1200 BRKINT IGNPAR
ISTRIP ICRNL OPOST ONLCR CS7 PARENB CREAD ISIG ICANON
ECHO ECHOK IXANY TAB3 #login: #300
```

After modifying **gettydefs**, we recommend that you run `getty -c` on the new **gettydefs** to verify that you have set it up correctly. `Getty -c` reports numbers for the following:

- basic terminal input — keyboard to computer — control (iflag);
- output — computer to screen — control (oflag);
- hardware control (cflag), and
- line discipline (lflag).

You can compare your numbers with the ones in **termio**(7) to determine if you have things set up the way you want them.

You can use **stty** for setting characteristics of serial ports connected to devices such as line printers; this procedure is much like using it for login ports. The basic idea is to match the **stty** settings with the printer's settings. In this case, however, **getty** does not set up the **termio** for the port. See the next section for more information.

4.11 Connecting a Serial Printer

Serial printers are TTY devices, from the Opus5 point of view. Thus, connecting a serial printer is like adding a terminal (see previous section), except it's simpler. You don't need to worry about **getty** and **gettydefs**, since your printer port is not a login port.

Follow the procedure outlined in this section to add a printer; if you encounter difficulties, the previous section might contain helpful background information.

1. Connect the serial printer to one of the PC serial ports. A null modem is required, because the standard serial ports on the PC are designed to be connected to communications devices (e.g., modems). A null modem is a device that reverses the pin configuration so that the

port can be connected to devices other than communications devices.

2. Establish the required port characteristics for your printer. You can do this with the command **stty**(1). The basic idea is to match the **stty** settings with the printer's settings. We can provide only general guidelines for this, since different printers require different settings. You must use the vendor documentation supplied with your printer to determine how to modify it. This might require some experimentation. The manual entry **termio**(7) describes the various possible settings and what they do.

   You might wish to use the special devices **/dev/tty128**, **/dev/tty129**, etc., for printer ports. See Section 5.3 for further information.

   Whenever a serial port is not open by any process, its **termio** structure is reset to the following:

```
c_iflag = 0

c_oflag = 0

c_cflag = B300 | CS8 | CREAD | HUPCL

c_cflag = 0

c_line = 0

c_cc    [8] = {   '\0177'/* delete */
             '\034'   /* control `|' */
          '#'
          '0'
          '\04' /* control D */
          '\0'
          '\0'
          '\0'
            }
```

   These are the default settings that **stty** finds in the **termio** structure; thus, if you want these settings, you do not have to reset them with **stty**.

3. When you issue an **stty** command on a port, it opens the port. But unless the port has been opened by *another* process, the **stty** settings will be lost as soon as the **stty** command is complete. This is because the structure **termio** is reset whenever a device is closed, and **stty** closes the device when the **stty** command is done. Thus, if you do not ensure that the port remains open by some other means, and you issue an **stty** command to, say, set up your printer, and then do your **lp** command, the settings you established with **stty** will *not* have taken effect.

   One way to ensure that the device remains open is by issuing a **sleep**(1) command to the port in background mode while you do the **stty** commands and your other commands. For example,

```
$ sleep 1000000000 < /dev/tty128 &
$ stty sane < /dev/tty128
$ cat file > /dev/tty128
```

   Remember that you must use the port during the time the **sleep** is running; the port is closed when the **sleep** terminates. Also, if you do not want the **sleep** to be terminated when you log out, you must precede the command with **nohup**(1), as follows:

```
$ nohup sleep 1000000000 < /dev/tty128 &
```

4. NOTE: **Stty** does not work on parallel ports (e.g., the **lp** device).

5. Once you have set up the printer, you can use it as standard output of ordinary Opus5 commands such as **cat**(1) and **pr**(1), as follows:

```
$ pr file > /dev/tty128
$ cat file > /dev/tty128
 etc.
```

The System V **lp** spooling system must be installed as a separate procedure (see next section) before **lp** will work with the new device.

4.12 Setting Up the Line Printer Spooler

This section summarizes the steps involved in setting up the System V line printer spooler. For more complete information, see the document "LP Spooling System" in the *UNIX System V Administrator Guide* (AT&T Publication Number 307-101).

1. Ensure that the printer is properly connected. If it's a serial printer, ensure that it has the correct **stty** settings. You can verify these things by issuing a **cat**(1) or **pr**(1) command and redirecting the output to the printer. For example, if the printer is the device **/dev/lp**, issue the command:

   ```
   # cat /etc/passwd > /dev/lp
   ```

   Do not proceed until this works correctly.

2. Most of the **lp** administration commands are in the **/usr/lib** directory. This path is usually not part of the PATH environment variable, so move to that directory.

   ```
   # cd /usr/lib
   ```

3. Make sure the **lp** scheduler is not running. Some of the **lp** administration commands do not work while the scheduler is active.

   ```
   # lpshut
   ```

4. Remove the scheduler lock file if it exists.

   ```
   # rm /usr/spool/lp/SCHEDLOCK
   ```

5. Initialize the spooler logical and physical device names. The "-p" option specifies an arbitrary name to be used for the printer in **lp** messages. The "-v" option specifies the actual device name. Also specify the driver model to use. This can be changed later. We suggest you start with *dumb* just to get the spooler going, and modify the model to suit your exact specifications later. This example assumes you want to call your printer "lp".

```
# lpadmin -plp -v/dev/lp -mdumb
```

6. Define the system default device.

```
# lpadmin -dlp
```

7. Allow the spooler to accept requests going to the logical printer named "lp".

```
# accept lp
```

8. Start the **lp** scheduler. This creates a process that handles all **lp** requests. It appears as "lpsched" when you do a **ps -ef** command.

```
# lpsched
```

9. Allow the logical printer "lp" to print jobs queued by the spooler.

```
# enable lp
```

10. Add the following lines to the file **/etc/rc**. This will start the scheduler each time the system is brought to multi-user mode. Add these lines after the line "echo cron started".

```
rm -f /usr/spool/lp/SCHEDLOCK
/usr/lib/lpsched
echo "lp spooler started"
```

11. The **lp** spooling system should be enabled and ready to print now. To check the status, type "lpstat -plp". You should see a response similar to the following:

```
printer lp is idle.  enabled since Jan 01 12:00
```

You can print files in at least four ways. **Lp** uses

/usr/bin/banner to write a header page, so make sure this program is installed. Then try one of the following examples:

```
# lp /etc/motd
# lp < /etc/motd
# cat /etc/motd | lp
# lp -c /etc/motd
```

The last three examples make a copy of the file for **lp**. The first example uses the file directly, so any modifications made while the file is printing may cause the output to be a combination of the two versions.

12. Modify your model program as required. Examine the files in the directory **/usr/spool/lp/model** for sample model programs. They are all simple shell scripts. When you have found or modified one to suit you, shut down the scheduler using **lpshut**, remove SCHEDLOCK, and re-issue the **lpadmin** command, this time specifying the name of the model you have chosen.

4.13 Changing the Maximum File Size

The maximum file size in Opus5 is controlled by the **ulimit**(2) system call. The default limit is 2048 blocks of 512 bytes each, or about 1 Mbyte.

The superuser can change the maximum file size directly from the shell, using `ulimit -f`. See **sh**(1). This option works only for root's processes, however. This section outlines a strategy for increasing the maximum file size for selected ordinary users. The strategy is, in summary, to write a C program that does a **ulimit** system call and exec's a shell; then substitute this program for each ordinary user's login shell in **/etc/passwd**.

The following C program, which we have called **unlimit.c**, changes the maximum file size to 20000 blocks, or about 10 Mbytes. You can substitute whatever number you like for "NEWLIMIT" in the program.

```
/* unlimit.c */
/* C program to change maximum file size in Opus5 */

#include <stdio.h>

#define NEWLIMIT20000

long ulimit();

main(argc,argv,envp)
char **argv;
char **envp;
{   if (ulimit(2,NEWLIMIT) == -1)
        fprintf(stderr, "can't change max file size\n");
    setuid(getuid());
    execve("/bin/sh",argv,envp);
}
```

After inputting this program, do the following steps:

1. Compile it as follows:

   ```
   cc unlimit.c -o unlimit
   ```

2. Log in as root, and issue the following commands:

   ```
   # mv unlimit /opus/bin/unlimit
   # chown root /opus/bin/unlimit
   # chmod 4755 /opus/bin/unlimit
   ```

3. Edit the file **/etc/passwd**, adding "/opus/bin/unlimit" to the end of the line for each user permitted to have the larger file size. For example, if you want to give the larger file size to the user terry, change terry's line from

   ```
   terry:<passwd>:<user #>:<group>:<id>:<home_dir>:
   ```

   to

```
terry:<passwd>: ... :<home_dir>:/opus/bin/unlimit
```

This completes the procedure for changing the maximum file size for selected users.

4.14 Backup and Restore

This section describes how to backup Opus release diskettes, and how to backup and restore Opus5 file systems and files.

4.14.1 Backing Up Opus Release Diskettes

Use the DOS command **diskcopy** to backup your Opus release diskettes; use the DOS command **diskcomp** to verify the copy. See DOS documentation for further information.

4.14.2 Backing Up Opus5 File Systems and Files

There are several methods you can use to backup and restore Opus5 file systems and files. All involve copying to floppy diskettes. Each method is discussed in separate sections below. The methods are:

- Use **cpio**(1) to the raw floppy device (recommended method).
- Use the DOS **backup** facility (for whole DOS-file-based virtual drives only).
- Use any other standard **UNIX** method for copying files to tape, such as **tar**(1), **finc**(1M), **frec**(1M), **dd**(1), etc., and use the raw floppy device instead of a tape device.

4.14.2.1 Using cpio

The commands **find**(1) and **cpio**(1) allow you to make multiple-diskette backups and also retrieve individual files. You can either position yourself in the root directory (/) and copy everything in Opus5; or position yourself in some subdirectory and copy everything in the tree beneath that point.

For example, to copy all of Opus5 to a set of diskettes, type:

```
# find / -print | cpio -ocB > /dev/rflpa
```

To copy only a portion of the hierarchy, say from **/usr/bin** down, type:

```
# find /usr/bin -print | cpio -ocB > /dev/rflpa
```

When a new diskette is needed, **cpio** gives an error message and prompts for the name of the backup device; respond with "**/dev/rflpa**".

To restore an entire diskette or set of diskettes created with **find** and **cpio**, assuming the diskettes contain a backup of all files in the directory hierarchy from **/usr/bin** down, type:

```
# cd /usr/bin
# cpio -icvdumB < /dev/rflpa
```

To restore a single file, mount the appropriate diskette and type:

```
# cpio -icvdumB < /dev/rflpa <filename>
```

To read a **cpio**-format diskettes to determine its contents, mount the diskette and type:

```
# cpio -icvt < /dev/rflpa
```

In copying a long file, **cpio** might split the file across two or more diskettes. To restore such a file, all diskettes in the backup set must be loaded in order. Issue the command to restore a single file. When you reach the diskette containing the first part of the file, **cpio** will copy the first part of the file and then prompt for the filename and the location of the rest of the file. Mount the next diskette and respond with the filename and the destination (**/dev/rflpa**).

Both Opus5 and DOS view the sets of files created by **cpio** as discrete physical tape volumes. DOS names them **opfil** files; and if you do a DOS **dir** command on a diskette containing **cpio**-created files, this is the name you will see (instead of the Opus5 filenames). To list the files on the diskette, use **cpio** with the regular "`-itv`" option.

4.14.2.2 Using the DOS Backup Facility

It is possible to have file systems that are, from DOS' point of view, simply large DOS files. You have these if you install Opus5 by the method described in Appendix A. Such files are much slower than partitions, and are not advised, although necessary for some BIOS-incompatible drives and controllers.

Such files are named, by default, `\opus\opfs\opfil0`, `\opus\opfs\opfil1`, etc. You can copy each whole file system to a DOS-formatted diskette (or set of diskettes) by using the DOS **backup** command, and then restore it using the DOS **restore** command. **Backup** makes an exact copy of every block in the file system, used or not.

A typical **backup** command is:

```
C>backup c:\opus\opfs\opfil0 a:
```

A typical **restore** command is:

```
C>restore a:\opus\opfs\opfil0 c:
```

See DOS documentation for other options.

This method is desirable if you are concerned about restoring the entire file system, rather than pieces of it; this is not a good method if you expect to use the backup diskettes to restore individual Opus5 files.

This method does not work for partition-based virtual drives, the standard Opus implementation.

4.14.2.3 Other Methods

UNIX systems provide several other methods of transferring data from disk to tape, e.g., **tar**(1), **dd**(1), **finc**(1M), and **frec**(1M). If you prefer, you can use these programs for backup and restore by substituting **/dev/rflpa** for the name of the tape device. See the AT&T documentation for further information on these programs.

Both Opus5 and DOS view the sets of files created by **tar** and other tape utilities as discrete physical tape volumes. DOS names them **opfil** files; and if you do a DOS **dir** command on

a diskette containing **tar**-created files, this is the name you will see (instead of the Opus5 filenames). Use regular options of Opus5 commands (e.g., "`tar tv ...`") for listing the contents of the diskette.

4.15 Shutting Down Opus5

In a multiuser environment the best way to shut down Opus5 is the **shutdown**(1M) command. After running **shutdown**, issue the command **dos**(1*) with the "`QUIT`" option, and turn off power.

A safe and quick method when only one person is using the system is to use the Opus-supplied command **dos**(1*) with the "QUIT" option, which performs a **sync**(1) and returns to DOS:

```
$ dos QUIT

C>
```

5. OPUS ENHANCEMENTS TO SYSTEM V

Opus has added numerous features to System V:

- A shell-level interface to DOS, which allows you to do the following:
  - — execute DOS commands from the shell
  - — execute Opus5 commands from DOS
  - — transfer files between DOS and Opus5
  - — from the shell, get the return code of a program running in DOS (Opus5 Release C3 and later)
- A library of Opus-specific C-language functions. These provide an interface to DOS from within C programs.
- Local terminals.
- An enhanced line printer driver.
- An enhanced **dial**(3C) function.
- The Virtual Display Interface (VDI) and Computer Graphics Interface (CGI), which allow graphics capability.
- The ability to change UNIX system parameters using the program **unix**(1*).
- "Sessions" — The ability to have up to four simultaneous login sessions from the same physical PC console, as well as a fifth session in DOS. This feature is standard with Opus5 Release C3 and later releases.
- A DOS program **opdisk**, which allows you to use large-capacity disks in your PC. This feature is standard with Opus5 Release C3 and later releases.

Each of these features is described in a separate section below.

5.1 Shell-Level Interface to DOS

This section describes the kinds of shell-level interaction between Opus5 and DOS. It has three parts:

1. DOS Commands from Opus5 — This section describes the DOS commands that are directly available from the Opus5 shell, and also describes a mechanism for executing any DOS command as a shell procedure.
2. Opus5 Commands from DOS — This section describes how to execute Opus5 commands from within DOS by using either the **opunix**(1*) interface or DOS batch files.
3. Transferring Files between UNIX and DOS — This section summarizes special commands supplied by Opus for transferring files to and from DOS and Opus5.
4. DOS Return Code — This section describes how to get, from the UNIX shell, the return code of a program running in DOS.

The "Sessions" feature also provides an interface to DOS, through the DOS window. Sessions may be more suitable than the DOS interface described here for certain applications. See Section 5.9 for details on Sessions.

As a protection, Opus has designed the DOS interface (and Sessions) to be fully accessible only to the console user. Note that the DOS interface is severely restricted from TTY ports. This is because the DOS interface can suspend UNIX; by default, if one user forces the console to go to DOS, all other users are suspended.

See the Opus manual pages on **opmon**(1*), **udcp**(1*), **ducp**(1*), **opdos**(1*), and **dos**(1*) for more information on the commands and facilities described in this chapter.

5.1.1 DOS Commands from Opus5

From the console, you can issue any DOS command directly from the UNIX shell, as shown in the examples below. You can also optionally continue to run UNIX in background mode while you are in DOS.

NOTE: If other users are logged in on TTY ports, be sure to always use the `BG` option when you go to DOS. Otherwise, their UNIX processes will be suspended while you are in

DOS.

To issue any single-line DOS command from Opus5, type:

```
$ dos [BG] DOS command
```

The following restrictions apply:

- For DOS releases earlier than 3.0, you cannot specify the DOS command using a full DOS pathname; therefore, the DOS command must be either in the current DOS directory, or in your DOS PATH.
- The DOS filename cannot contain any shell special characters unless they are quoted.

The DOS command you specify is issued, and you return to the Opus5 shell when it is complete. For example, to check your DOS disk while Opus5 is running, you can type:

```
$ dos chkdsk
```

To return to DOS temporarily from Opus5, issue the command:

```
$ dos
```
or
```
$ dos BG
```

This puts you into DOS until you issue a **unix** command from DOS. The **unix** command normally resides in the DOS directory `\opus`, so you must adjust your DOS PATH to include `\opus`, or change directory to `\opus` before issuing the **unix** command, or copy **unix.bat** and **opmon.exe** to a directory in your DOS PATH.

Any one-line DOS command can be turned into a shell procedure by linking it to the file **/opus/bin/dodos**. This allows you, from the console, to issue the DOS command just like any other Opus5 command. For example, to link the DOS command **chkdsk** to **/opus/bin/dodos**, type:

```
$ ln /opus/bin/dodos /opus/bin/chkdsk
```

Then you can issue the **chkdsk** command from the shell

prompt:

```
$ chkdsk c:
```

The **chkdsk** is executed and you are returned to the shell.

The DOS interface is restricted from TTY ports. The TTY port does not, as a rule, have access to DOS. For example, you can switch the system from UNIX to DOS from a TTY port by typing:

```
$ dos REMOTE
```

However, the DOS prompt will appear on the *console*, and interaction with DOS must take place there. At the TTY, UNIX will simply be suspended.

Sometimes it make sense to do a `dos` command from a TTY port using both the `BG` and `REMOTE` options, e.g., to run an application under DOS that sends output to the DOS COM port.

More complicated DOS interfaces can be programmed using **opdos**(1*). Use **/opus/bin/dodos** as an example.

If the DOS command is a DOS batch file, it must end with a **unix**.

5.1.2 Opus5 Commands from DOS

There are two ways to execute UNIX commands (including shell scripts) from DOS. Both approaches work at the console only.

1. One way, **opunix**(1*), works similarly to the **dos** command from UNIX. The **opunix** approach can be used only when you have gone to DOS with a `dos BG` command from UNIX. It uses the **udio** channel and requires that the **udiodaemon**(1*) be running. This approach is discussed in Section 5.1.2.1.

2. The other way uses DOS batch files that write the UNIX command into the file **\opus\oparg** and then exit. This approach can be used any time; it does not require you to use `dos BG`. It is discussed in Section 5.1.2.2.

5.1.2.1 The opunix Approach

UNIX commands can be executed from DOS via the **udio** interface, with the **opunix** command. Using this interface, DOS and UNIX run simultaneously. To use **opunix**, you must have entered DOS via a `dos BG` command on the UNIX side.

The **udio** driver generates a bi-directional communications channel. **Udiodaemon** talks to this channel from the UNIX side, while **opunix** talks to it from DOS. **Udiodaemon** uses the Opus-supplied device **/dev/udio**. **Opunix** uses the loadable driver **dosudio.sys**, which is in the **\opus** directory under DOS.

To set up the **udio** interface, perform the following steps:

1. Verify that the DOS file **\opus\opus.cfg** contains an entry like the following:

   ```
   <udio=udio>
   ```

   If there is no such entry, add one.

   Two modifications to this line improve **opunix**'s character output:

   a. You can optionally specify a free software interrupt vector using the `int=` option, as shown in the example below:

      ```
      <udio=udio(int=0x58)>
      ```

      See the *Technical Reference Manual* for your system for further details.

   b. If you have a free hardware interrupt vector, you can improve *udio* performance by running the Opus board using interrupts. For example, to use hardware interrupt vector 7, type:

      ```
      [i=7]
      ```

      See Chapter 8 for guidelines on choosing hardware interrupts for the Opus system. Note that you might need to change hardware switch settings.

2. Add the following entry to the DOS file **\config.sys**.

```
device=\opus\dosudio.sys
```

3. Cycle power in the PC in order to install the loadable driver.

4. Edit the file **/etc/rc**, and add the following lines. You can add them just after the call to **/etc/cron**.

```
/opus/bin/udiodaemon
echo udiodaemon started
```

5. Re-boot UNIX and go to multi-user mode (`init 2`). The **udio** interface should now be ready to use.

To use the *udio* interface, login to UNIX at the console, and enter DOS from UNIX by typing the following:

```
# dos BG
```

This puts you in DOS. Currently running UNIX commands, except those that use or manipulate DOS files, continue to run in background. From DOS you can now use the *opunix* command to execute UNIX commands from DOS. See *opunix*(1*) for more information and examples.

5.1.2.2 The Oparg Approach

We have seen how once Opus5 is started up, you can go to DOS using the **dos** command, and go back to Opus5 with the command **unix**. Conversely, from the DOS perspective, once Opus5 is started and you have gone to DOS, you can return to Opus5 with the **unix** command and issue **dos** commands from Opus5 as much as you like. You can also write DOS batch files that resume Opus5, execute a single UNIX command (which can be a shell script), return to DOS, and optionally execute a DOS command.

For example, suppose you want to be able to list the contents of the current Opus5 directory without leaving DOS. You can create a DOS file called, say, **ls**, that executes the Opus5 **ls**(1) command and returns to DOS by spawning a new DOS shell. The DOS file would look like Figure 5-1:

```
echo off
echo ls %1 %2 %3 %4 %5 %6 %7 %8 %9 > \opus\oparg
exit
```

Figure 5-1. Sample **ls** Command Batch File

Put the Opus5 command you want to execute in the place of "ls" in the example. "%1 %2", etc. are DOS command line variables, which are substituted for by the arguments you give the DOS **ls** command. UNIX command line options such as "-l" or "-a" can be substituted for % options, in addition to filenames and directory names. Other UNIX commands can also be invoked using pipes and redirection, as long as the number of command line arguments is not exceeded. This enables you to issue all the following commands from DOS:

```
C>ls
C>ls -l
C>ls -la
C>ls -la filename ...
C>ls -la filename dirname ...
C>ls -l > files
C>ls -l "> files"
```

Double quotes around symbols and filenames preserve redirection or piping within operating systems. For example, the command `ls -l > files` puts the result of the `ls` command into the DOS file `files`. The command `ls -l "> files"` puts the result of the `ls` command into the UNIX file `files` in the root directory.

In summary, if Opus5 has been booted, and you have issued the **dos** command, then your new DOS **ls** command resumes Opus5, executes the Opus5 **ls** command, and then returns to DOS by spawning a new DOS shell. Note that this only works if you have booted Opus5 and reached DOS via the **dos** command from Opus5.

Remember that your Opus5 command can be a shell script, so you are not really limited to one Opus5 command using this mechanism.

Complicated DOS batch files might require that you resume DOS in the *same shell*. This can be accomplished by coding the DOS batch file in a slightly different way, as shown in Figure 5-2.

```
rem file is called abc.bat
if %1 = x1 goto label1
      :
      .
echo off
echo DOS abc x1 ls %1 %2 > \opus\oparg
exit
:label1
      :
      .
```

Figure 5-2. Sample Batch File Returning to Same DOS Shell

The keyword "DOS" in the file **\opus\oparg** tells **opmon** to take the next two arguments as (1) the name of a DOS file to be executed when DOS resumes, and (2) an argument to the DOS filename. In this example, called `abc.bat`, the command `abc x1` is executed when DOS resumes; this picks up the batch file where it left off when it did the UNIX command.

See DOS documentation for further information on DOS batch files.

5.1.3 Transferring Files Between UNIX and DOS

Opus provides the following special commands for the PC environment:

- **ducp**(1*) DOS to Opus5 copy
- **udcp**(1*) Opus5 to DOS copy

Ducp and **udcp** are modeled on the UNIX **cp**(1) command, with certain restrictions, as noted in the manual pages. As of Opus5 Release C3, they take "wild card" file specifications.

See the appropriate manual pages for complete information.

5.1.4 DOS Return Code

The DOS interface gives you a straightforward way to get the return code of a `.exe` or `.com` program running in DOS from the UNIX shell or a program running in UNIX. Just type one of the following:

```
$ dos EXE [doscmd.exe]
```

or

```
$ dos COM [doscmd.com]
```

according to whether the DOS command you want to execute is a `.exe` or a `.com` command. Then, if you type the following from the shell:

```
$ echo $?
```

the return code of the DOS program is printed. You can also use these sequences within shell scripts.

This facility does not work to get the return code of a DOS `.bat` file.

If you want to know why this works, read on.

5.1.4.1 Why This Works

To explain this enhancement, we must review how DOS commands are executed from UNIX. When you run a DOS program using **/opus/bin/dos**, the Opus program **/opus/bin/opdos** puts the DOS command in the file `oparg` on the DOS side. It then instructs **opmon** to run the DOS

program with pause code 2. Pause code 2 is an Opus convention, which tells **opmon** to fork a new DOS shell with the contents of `oparg` as its argument. The new DOS shell runs the DOS program and then exits. The return code of the DOS shell, rather than the return code of the program, is sent to UNIX.

Note that this is not analogous to the way shell scripts work in UNIX. If a UNIX shell script explicitly specifies an exit code within the script (e.g., `exit 4`), then that exit code is returned to the calling program. But if the shell script does not explicitly specify an exit code, the exit code returned is that of the last program run by the shell script. If DOS shells worked this way, there would be no problem.

However, if a DOS program could be run from UNIX *without* invoking a new DOS shell, then the program's return code could be sent directly to UNIX. The Opus pause code 3 allows just this. If a DOS program is run from UNIX with pause code 3, it is run directly in DOS, without a new DOS shell. Thus, its return code is sent to UNIX.

Opus chose to make pause code 2 the default, because of an important restriction on code 3. Code 3 can be used *only* with `.exe` or `.com` programs; it *cannot* be used with DOS batch files. Code 2, on the other hand, can be used with anything. The enhancement to **/opus/bin/opdos** included in this release, then, works only for `.exe` or `.com` programs.

See the file **/opus/bin/dos** for implementation details.

5.2 Opus Library

Opus provides a specialized C language library, which includes an interface to DOS functions, and other functions. The library is in **/usr/lib/libopus.a**. It is documented in Appendix F, and comprises all the pages labeled "3*". **/usr/lib/libopus.a** includes the following functions:

doschdir	Change the current DOS directory.
doschdrv	Change the current default DOS drive.

doschmod	Change or report the attributes of a DOS file.
dosclose	Close a DOS file.
doscreate	Create a DOS file.
dosdiskfree	Report current DOS disk usage.
dosioctl	I/O control for DOS files.
dosinit	Open the DOS device.
doslseek	Position the read/write pointer within a DOS file.
dosmkdir	Create a DOS directory.
dosopen	Open a DOS file.
dosread	Read a DOS file.
dosrename	Rename a DOS file.
dosrmdir	Remove a DOS directory.
dosunlink	Delete a DOS file.
doswrite	Write a DOS file.
syscmdline	Get the command line used to invoke the current **opmon**.
sysexit	Exit to DOS.
syspause	Pause to DOS.
sysreadio	Read the contents of the PC bus I/O port.
sysreadmem	Read the contents of the PC bus memory.
syswriteio	Write a value into the PC bus I/O port.
syswritemem	Write a value into the PC bus I/O memory.

5.2.1 Notes for Using Opus Library

Keep in mind the following when using the Opus library:

1. You must declare *doserrno* in order to receive complete error reporting. Thus, you should insert the following

line in any program using DOS function calls.

```
extern int doserrno;
```

2. DOS imposes a limit on the number of files that can be open at one time. Remember to close files when you are finished with them, so you won't run out of file handles.
3. If your Opus5 file systems are DOS files, use handles with caution. The file systems are opened by **opmon** and have file handles just like other DOS files. If you issue a **dosclose** function using the wrong handle, and the handle you use happens to equal the handle of an Opus5 file system, the file system will crash.
4. DOS files are not automatically closed when your program exits; you might need to handle signals causing termination.

5.3 Local Terminals

Opus provides two devices, **/dev/tty128** and **/dev/tty129**, which are not part of the standard AT&T release. These devices are just like **/dev/tty0** and **/dev/tty1** (in fact, they refer to the same physical devices), except that **/dev/tty128** and **/dev/tty129** can be opened without Carrier Detect present.

These devices exist to solve two dilemmas.

The First Dilemma

Suppose you have a talented modem that can both receive calls and also dial out. You want other people to be able to log in to your machine using this modem. So you've put an entry in **/etc/inittab** which starts a **getty** process for one COM line, to enable logins. **Getty** must succeed in opening the port in order to send out its `login:` message. The open call works only if the DCD (Data Carrier Detect) signal is active. In other words, **getty**'s open call is *blocked* until DCD is active; **getty** just sits and waits until this happens. The DCD signal is active when carrier is noted on the line. This all works smoothly for purposes of *receiving* calls: the caller initiates carrier, which raises DCD, which causes **getty**'s open to

succeed, which causes a `login:` prompt to be sent to the port, which allows your caller to log in.

But suppose you would also like to use this very same modem to dial out. Suppose you try now to use **cu**(1) or **uucp**(1) to dial out. These programs also must open the port. But, as we saw above, *open* works only if DCD is high, and DCD is high only when there is already carrier on the line. Thus, programs like **cu** and **uucp**, which want to call out, can never successfully open the port. This is a true dilemma: the circumstances under which you might call out (port opened) are exactly the circumstances that make it impossible for you to call out (if the port is opened, DCD is high, which means the port is in use)!

Because of this dilemma, most UNIX systems recommend that you have two separate lines and modems if you want both dial-in and dial-out capabilities.

The devices **/dev/tty128** and **/dev/tty129** were born in response to this problem. When the serial driver sees a call to one of these devices, it allows the line to be opened even if DCD is not present. **Tty128** refers to the same physical device as **tty0**; **tty129** refers to the same physical device as **tty1**. You just use **tty0** or **tty1** for dialing in, **tty128** or **tty129** for dialing out.

The new devices have the same major device number (9) as **/dev/tty0** and **/dev/tty1**; that is, they are controlled by the same driver. Their minor device numbers are derived by adding 128 to the minor device numbers of the regular serial ports they are associated with: thus, the device **tty128** (associated with **tty0**) has minor device number 128; **tty129** (associated with **tty1**) has minor device number 129. In general, in the Opus implementation, any TTY device whose minor device number is derived by adding 128 to an already-existing device will have the special properties of **tty128** and **tty129**.

The Second Dilemma

But now there is a second dilemma. We saw how **getty** patiently waits for DCD to get high so it can open the port and send out its `login:` invitation. Suppose you succeed in opening the newly-created **/dev/tty128** or **/dev/tty129**, and you

get the modem to connect to another modem. This makes DCD high! **Getty** has been blocked just waiting for this to happen. (**Getty** doesn't care *how* this happened.) So **getty** will get busy trying to send a `login:` message out onto the line. If you are going to control the port, **getty** must be stopped.

The serial driver knows about this threat. When it opens devices **/dev/tty128** and **/dev/tty129**, it automatically checks for any processes blocked waiting for DCD on the associated serial ports. That is, when **tty128** is opened, it looks for processes blocked waiting for DCD on **tty0**; and when **tty129** is opened, it looks for processes blocked waiting for DCD on **tty1**. The driver then suspends all activity on the associated port; the effect is that the **getty** simply continues to be blocked as long as **tty128** or **tty129** are open.

With both of these problems handled in Opus software, you can use your modem for both dial-in and dial-out. Just use **tty0** or **tty1** as dial-in lines (put entries in **/etc/inittab** for them) and **tty128** and **tty129** as dial-out lines (no entries in **inittab**).

Sp far we have talked about **tty128** and **tty129** as modem ports only. The next paragraphs talks about their use as terminal ports.

Terminal Ports?

The special TTY devices **tty128** and **tty129** should be used as terminal ports only if special care is taken. You might wonder at this. **Getty** opens terminal ports, and it doesn't care about carrier, so you might think it would always want to ignore DCD, so **tty128** and **tty129** would be perfect. This is theoretically true, and often true in practice as well — but not always.

When a port is being used as a modem, DCD is always hooked up, because it is essential for successful communication using modems. But sometimes when people use ports as terminals, and especially when make their own null modem cables, they reason as follows: "If I use **tty0** or **tty1** I must ensure that DCD is high; otherwise, I can't open those ports. But terminals don't care about DCD anyway.

Why don't I just save myself some soldering and leave DCD unconnected? Then I can use those special **tty128** and **tty129** ports, which can be opened whether or not DCD is present."

Unfortunately, not hooking up DCD (or DSR or CTS) can create a problem, because all the RS232 signals are electronically sensitive. If DCD is not hooked to a solid voltage, either to another wire or to ground, it receives a lot of noise. The noise causes the signal to jump up and down very quickly, electronically, and **opmon** notices every time it changes state. When **opmon** notices this, it interrupts UNIX, and the serial driver has to pay attention, if only to ignore the signal. This uses up lots of CPU cycles, and the system can run noticeably slower. In fact, the kernel logs you off if you try to use **tty128** or **tty129** without either connecting DCD or using the `local` option.

Therefore, if you want to use **tty128** as a terminal port, you must choose one of the following two options:

1. You can make sure DCD, CTS, and DSR are all physically connected. Store-bought cables generally hook up every signal, so if you are using them, you probably don't have to worry about this problem; you can go ahead and use **tty128** or **tty129** as a terminal. If you are using home-made cables, you can solder or otherwise connect these signals yourself.

   OR,

2. You can use use a special option for TTY devices in **opus.cfg**. That option, `local`, makes **opmon** ignore the DCE's state changes, so UNIX is not interrupted. The **opus.cfg** entry looks like this:

   ```
   <tty0=com1(local)>
   ```

We recommend option (2) because it's easier. You modify the **opus.cfg** entries for **tty0** and **tty1** because there are no **opus.cfg** entries for **tty128** and **tty129**. (Remember that the driver knows they are the same physical devices as **tty0** and **tty1**.)

Special devices such as **/dev/tty128** and **/dev/tty129** represent only one approach to a solution to the problems mentioned here. Standard System V software provides a different approach.

You can open a regular TTY device even if no carrier is present by using **open** with the O_NDELAY flag set. Then using **fcntl**, turn off the O_NDELAY flag. Then do an **ioctl**(2) call on the device and set the flag CLOCAL. This flag tells UNIX that no modem control applies to this device. All the while, **getty** remains undisturbed and continues to be blocked.

The standard **dial**(3C) routine knows about this strategy, and uses it when you call **cu** with the "-m" option.

5.4 Line Printer Driver

The Opus **lp** driver is similar to **lp**(7), with the following exceptions:

- Opus **lp** provides an interface to the parallel port(s) on the PC, rather than to the DEC LP-11 UNIBUS line printer.
- Minor **lp** device numbers correspond to **opmon** drivers as follows:

Minor device number	**opmon** driver
0	`<lpt1>`
1	`<lpt2>`
2	`<lpt3>`

- Adding 16 to the minor device number causes the driver to use half-ASCII mode (64-character set).
- Adding 32 to the minor device number causes the driver not to automatically wrap long lines that extend past the current line length.
- Adding 64 to the minor device number causes the driver to pass all characters through with no conversion or modification. This is useful for programs that use printers for graphics output.

- The standard **lp** driver truncates lines longer than 132 characters, and by default indents all lines 4 spaces. The Opus **lp** driver does neither, i.e., there is no limit on line length and no default indent.

5.5 Enhanced dial(3C) Function

Dialing programs by convention get information about dialers from the file **/usr/lib/uucp/L-devices**. The "call devices" field of the L-devices file specifies whether dialing is to be done by a Western Electric Automatic Call Unit (ACU) or whether the connection is direct. The standard **dial**(3C) procedure looks at the "call device" field of the L-devices file and dials the phone if an ACU device is specified. Only the two options, ACU and direct, are permitted as "call devices" in the L-devices file; and the standard **dial** procedure handles ACU devices only. Thus, the standard system provides no straightforward way to allow dialing by other equipment (for example, modems with dialing capability).

Opus has generalized the **dial** procedure to permit dialing by devices other than ACUs, such as modems. The **dial** function can then be invoked from any user program that wishes to dial a phone via an ordinary modem. The Opus modification permits you to supply the name of your own customized dialing program in the "call devices" field of **/usr/lib/uucp/L-devices**. The enhanced **dial** procedure then recognizes your program name and, after the communications line has been opened, **exec**'s your dialer program. See the Opus manual page **dial**(3C*) for more information and sample usage.

Opus provides source and object for a sample customized dialing program, which works for Hayes and Hayes-compatible modems. The source is **/opus/src/hayes.c**; the object is **/opus/bin/hayes**.

5.6 VDI Interface

Opus5 provides an interface from Opus5 to the Graphics Development Toolkit, IBM's implementation of the Virtual Device Interface (VDI). (In this document, we use the terms

"Graphics Development Toolkit" and "VDI" synonymously.) VDI is a set of graphics library routines and device drivers. It resides in DOS, but is accessible from the shell. You can write programs that use these routines for text and graphics output on a variety of output devices.

5.6.1 Setting Up VDI

We assume that you have purchased and installed a compatible graphics card, and have the Graphics Development Toolkit software, including the 1.0 Update.

5.6.2 Installing VDI

This section lists the steps in installing the interface from Opus5 to VDI.

1. Install VDI according to the instructions that accompany the software. The basic steps are the following:
   - Copy the drivers and demo program to hard disk. (Support libraries are not required, unless you plan to do program development under DOS.)
   - Edit **config.sys** to load the drivers you require.
   - You may need to edit **autoexec.bat**, though this is usually not necessary; see VDI documentation for requirements.
   - Reboot DOS.
   - Verify that VDI works by running the VDI test program. (It may be necessary to type "cls" to restore the screen after the demo.) Do not proceed with the rest of this installation procedure until the demo runs.
2. Use **opconfig** to verify that the following device is listed in **opus.cfg**:

```
<vdi=vdi>
```

3. Boot Opus5 normally.

```
C>cd \opus
C>unix
```

5.6.3 VDI Usage Notes

In certain graphics modes, the IBM Enhanced Graphics Adapter uses memory segment A, which is the default Opus 100PM address. To eliminate this conflict, choose another available segment. For the Opus 108PM, reset switches 5-8 in the DIP switchpak at location D25. For the Opus 110PM, reset switches 7-10 of the DIP switchpak at location D2. If necessary, also modify the default segment using **opconfig**. See Section 8.2.3.

The functions **v_get_pels** and **v_put_pels** in the standard C binding are not available from Opus5. Opus provides two other functions, **v_get_npels** and **v_put_npels**, which are equivalent in functionality to **v_get_pels** and **v_put_pels**. The Opus-supplied functions require you to specify the number of 16-bit words in the pel_array. The syntax of the Opus-supplied functions is:

```
v_get_npels(device_handle, xy, pel_array, count);
v_put_npels(device_handle, xy, pel_array, count);
```

"Count" must be specified in 16-bit increments.

Opus provides three optional functions, **vdi_open**, **vdi_close**, and **vdi_flush**, in addition to those in the standard C binding.

vdi_open Controls two operations: whether or not each VDI procedure call is executed immediately; and the size of the buffer used. Depending on data, delayed (buffered) execution of VDI functions can enhance speed. If **vdi_open** is used, it must be prior to any VDI call. The syntax is:

```
vdi_open(imm_flag, buf, size, flag);
```

where

imm_flag is 1 for immediate execution of each call; 0 for buffered.

buf pointer to user-supplied output buffer. If "buf" is 0, the default buffer size (1024 bytes) is used.

size size in bytes of the desired output buffer.

flag Must be zero.

vdi_flush Outputs the contents of the output buffer even if the buffer is not filled. Syntax is:

```
vdi_flush();
```

vdi_close Called after the last VDI request to restore immediate execution and 1024-byte buffering. Allows you to re-specify a buffer size for output with another **vdi_open**. The syntax is:

```
vdi_close();
```

This function does a **vdi_flush**.

If the above functions are not used, the default buffer size is 1024 bytes and the output is flushed after every call. Note that the buffer size must be large enough to hold the data for any individual VDI call; that is, you cannot split data for a single call across buffers. Therefore, if your data for a single call is large (e.g., a large polyline call), be sure to change the buffer size using **vdi_open**, even if you want immediate execution.

The program **/opus/src/gannt.c** is the standard test program. You can use it to verify correct installation of the Opus5 portion of the VDI interface.

To compile a program using VDI functions, you must link it with the Opus5 VDI library. To do this, specify "-lvdi" on the compile command line. For example,

```
cc -o gannt gannt.c -lvdi
```

5.7 CGI Interface

Opus Systems provides an interface from Opus5 to the Computer Graphics Interface (CGI) package from Graphics Software Systems, Inc. This interface, which is called *GSS*CGI*, is a set of graphics library routines and device drivers. It resides in DOS, but is accessible from a UNIX application. You can write programs that use these routines for text and graphics output on a variety of output devices.

5.7.1 Setting UP CGI

We assume that you have purchased and installed a compatible graphics card, and have purchased the GSS*CGI software.

5.7.2 Installing CGI

The steps for installing the interface from Opus5 to GSS*CGI are the following:

1. Install GSS*CGI according to the instructions that accompany the software. The basic steps are the following:

   - Copy the drivers and demo program to the hard disk. Support libraries are not required, unless you plan to do application development under DOS. You will need to load a font directory with the font files corresponding to the driver(s) you are using. This directory, named **fonts**, is usually located in a subdirectory of the driver's.

   - Edit the DOS file **\config.sys** to load the drivers you require. The last driver specified should be

     ```
     device=\cgi\gsscgi.sys
     ```

     if your GSS*CGI files are in the `\cgi` directory. Make sure you include the font driver, as follows:

     ```
     device=\cgi\fontdrv.sys
     ```

- Make sure the DOS file **autoexec.bat** includes the following line:

```
set fonts=\cgi\fonts
```

- Run INSTFONT to install the fonts in your font directory. This will produce a new file named FONTLIST.DAT.
- Reboot DOS.
- Verify that DOS works with GSS*CGI by running the GSS demo program. Do not proceed with the rest of this installation procedure until the demo runs.

2. Using **opconfig**(1*), modify the Opus system configuration file **opus.cfg** to contain the following line:

```
<cgi=gsscgi>
```

3. Boot UNIX.

```
C>cd \opus
C>unix
```

5.7.3 CGI Usage Notes

In certain modes, the IBM Enhanced Graphics Adapter uses memory segment A, which is the default Opus 100PM segment. To eliminate this conflict, you must use another available segment, such as D. Change switches 5-8 of the Opus 108PM coprocessor (switches 7-10 of the Opus 110PM) in the DIP switchpak at location D25 (location D2 on the Opus 110PM). If necessary, use **opconfig**(1*) to modify the default segment listed in **opus.cfg**. See Section 8.2.3.

The program **/opus/src/cgannt.c** is the Opus test program for GSS*CGI. You can use it to verify correct installation of the Opus5 portion of the GSS*CGI interface.

To compile a program using GSS*CGI functions, you must link it with the Opus5 CGI library. Any file using the GSS*CGI interface must also contain the following line:

```
include <cgi.h>
```

To compile the test file, type:

```
# cc -o cgi_demo cgannt.c -lcgi
```

If you are converting an existing VDI program to run with GSS*CGI, you must do the following:

1. Add the CGI include file:

```
include <cgi.h>
```

2. Convert the following VDI function names to the corresponding GSS*CGI names:

VDI	CGI
vdi_open()	cgi_open()
vdi_flush()	cgi_flush()
vdi_close()	cgi_close()

5.8 The /unix Command

Opus provides a way for you to change important system parameters such as the number of system buffers, the system nodename, the swap area size and starting address, and the maximum file size, by options to the **/unix** command in the DOS file **\opus\unix.bat**.

Changing system parameters this way can be EXTREMELY DANGEROUS. You can do IRREVERSIBLE DAMAGE to your file systems if you make a mistake. Please use extreme caution.

For details, see the Opus manual page **unix**(1*).

5.9 Sessions

"Sessions" allows you to have up to four simultaneous login sessions from the same physical PC console. A fifth session can be in DOS. Simple keystrokes allow you to switch between sessions quickly.

The new devices are named **/dev/con1**, **/dev/con2**, and **/dev/con3**. The Sessions devices are available when you go to multi-user mode. Then, you can use the following default keystrokes to get access to different sessions:

alt-H	console
alt-J	con1
alt-K	con2
alt-L	con3
alt-D	DOS

The Sessions devices are **/dev/con1** through **/dev/con3**. These are created as part of regular C3 installation. The regular `console` device is installed by default with the special option `dos`, as shown below, to enable access to a DOS window with ALT-D from the regular console screen. The **opus.cfg** specifications for Sessions devices are thus as follows:

```
<console=console(dos)>
<con1=con1>
<con2=con2>
<con3=con3>
```

In UNIX, these devices use major device number 0 and minor device numbers 1 through 3.

In addition, the system file **/etc/inittab** automatically starts a **getty** process for each of these devices, so that by default a login prompt appears on each of them.

If you want fewer than four possible sessions, you can disable the unwanted ones by deleting or commenting out the appropriate lines in **/etc/inittab** and/or **opus.cfg**.

The following additional options are available:

1. Non-overlapping, variable-sized windows.
2. Access to DOS with one keystroke.
3. A "screen-saver" feature: the console blanks after a specified number of minutes.

These are available through the following options to the device specifications for the `console` and `con1-con3` devices in **opus.cfg**.

`bar=rownum` — The row where you want to place a horizontal bar. Such a bar is useful to divide the screen among sessions. For example, to place a bar at row 13 for `con1`, you would use the following:

```
<con1=con1(bar=13)>
```

`blank=minutes` — The number of minutes of inactivity before the physical terminal blanks out. For example,

```
<con1=con1(blank=20)>
```

`buf=num` — By default, **opmon** tries to figure out what kind of display adapter you have, so it can copy data in and out directly. By default, it does not use the BIOS. If you want it to use the BIOS to save and restore windows, use the following:

```
<console=console(buf=0)>
```

`dos` — Enable a fifth DOS session. Typing ALT-D puts you into DOS and puts UNIX in background mode (like `dos BG`). Unlike `dos BG`, no **sync**(1) is done. When in DOS, the diskette drive is disabled from UNIX, since UNIX is in the background. Type "`exit`" in DOS to put UNIX back in foreground.

This option comes with the standard C3 **opus.cfg** specification for `console`, shown below:

```
<console=console(dos)>
```

dos=keycode Use a different key to go to DOS. For sites that already use ALT-D for some other function. For example,

```
<console=console(dos=keycode)>
```

See the *IBM BASIC Manual*, Appendix G, for a complete list of keycodes.

key=keycode Use a different key to go to this session. For example, to change the key for `con1` to ALT-B, the **opus.cfg** specification for `con1` would read as follows:

```
<con1=con1(key=48)>
```

The number 48 is the key code for ALT-B. See the *IBM BASIC Manual*, Appendix G, for a complete list of keycodes.

page=pagenum **Opmon** ordinarily saves images of all sessions screens in memory buffers. However, some kinds of console hardware can store more than one 25 X 80 page of screen data in hardware buffers. If your hardware can do this, **opmon** doesn't need to save copies of all the screens. The `page` option lets you specify which hardware pages to associate with which sessions. For example, if you have two hardware pages, 0 and 1, you can associate page 0 with `console` and page 1 with `con1` by the following specifications:

```
<console=console(page=0)>
<con1=con1(page=1)>
```

rows=first-last The screen rows you want to set aside for a particular session. For example, to display `con1` output on the first 12 rows only, use the following:

```
<con1=con1(rows=1-12)>
```

The specifications **dos**, **blank**, and **buf** are global; that is, they can be specified once either for **console**, **con1**, **con2**, *or* **con3**, and they will affect all the sessions devices.

5.9.1 Sessions — Usage Hints and Examples

A number of regular UNIX commands tell you what session your screen is currently displaying. The most direct is **tty**(1). Just type the following:

```
$ tty
```

The name of the device you are using is returned.

The following sample **opus.cfg** lines for sessions devices allow you to go to DOS from any session, blank the screen after 20 minutes, and set up an equally-split screen for **con2** and **con3**:

```
<console=console(blank=20,dos)>
<con1=con1>
<con2=con2(rows=1-12,bar=13)>
<con3=con3(rows=14-25,bar=13)>
```

The popular editor **vi** by default expects a 25-line screen. Thus, if you use split screens, **vi**(1) will not work correctly within those screens unless you also modify your terminal definition. To do this, perform the following steps:

1. Edit the file **/opus/src/opus.ti** and add lines like the following. Note that these lines are specifically for a 12-line screen; modify appropriately if your screens will be larger or smaller.

   ```
   opus-pc-12,
         lines#12, use=opus-pc,
   ```

   These lines can be added anywhere in the file. The first line *must* start in the first column; the second line must start with a tab.

2. Compile **opus.ti** using the terminal database compiler **tic** as follows:

```
$ tic opus.ti
```

This puts a definition for `opus-pc-12` into the directory **/usr/lib/terminfo/o**.

3. Then when you login to the partial window, set your terminal definition to `opus-pc-12`, as follows:

```
$ TERM=opus-pc-12
```

Or, you can modify your **.profile** to accomodate all your sessions. The following example shows how to do this if `con2` and `con3` are partial windows, but `console` and `con1` are full windows:

```
TTY=`tty`; export TTY
TERM=opus-pc; export TERM
if [ $TTY = /dev/con2 -o $TTY = /dev/con3 ]; then
        TERM = opus-pc-12
fi
```

Finally, the sessions interface will become more natural to you if you remember the following general rules:

1. Use ALT-D to create and go to the DOS window.
2. Think of the DOS window as a fifth window, not a sub-process of a UNIX window. After you type ALT-D to create a DOS window, the DOS window has nothing to do with the UNIX session in which it was spawned. For example, if you are running in `console` and you type ALT-D to start a DOS window, you cannot access the DOS window with ALT-H, for example, even though that was where it was initiated. Access it using ALT-D only.
3. ALT-D puts DOS into foreground and UNIX into background. *UNIX remains in background mode until you type* `exit` *from the DOS window.* This means that processes running in UNIX continue to run, and you can initiate new processes, but you cannot access DOS files from UNIX. Remember that the DK driver uses DOS files for floppy access; this means that while UNIX is in background, the diskette drive cannot be accessed.

4. If you need to access DOS files or the floppy from UNIX, get out of ALT-D mode by typing `exit` from the DOS window.

5. If you use `dos BG` instead of ALT-D to initiate the DOS window, the UNIX session in which you started the `dos BG` is suspended and may appear hung. Type ALT-D to go to the DOS window. Type `exit` from the DOS window to restore the UNIX session.

5.10 Using Opdisk for Large-Capacity Disks

Opus5 release C3 includes the Opus program **opdisk**, which allows you to use some large-capacity disks on your PC. This section gives a step-by-step description of the procedures involved in adding a Priam 40- or 60-MByte disk to a PC, concluding with instructions for using **opdisk**. For illustration purposes, we assume you have an IBM PC AT with an Opus board installed, and a Priam InnerSpace disk (not installed). We do not guarantee these instructions for other configurations, although it is possible to use other large disks on other systems as well. The Opus manual *Adding Large Disks to the IBM PC AT or XT* describes in a more general way the various problems and solutions for adding large disks to the PC. Contact Opus for a copy of this document.

The basic steps are listed below; each step is discussed in more detail in the sections that follow.

1. Get a Priam Innerspace disk.

2. Physically install the Priam disk.

3. Run the AT SETUP program to enable use of the Priam disk.

4. Determine if you want a DOS partition on the Priam disk. If so, use the DOS FDISK program to create a DOS partition on the Priam disk.

5. Modify the DOS file **config.sys** to include the loadable driver **opdisk**.

6. Use Opus software to create a partition for UNIX and install UNIX.

You should have all the following additional documentation:

1. *Opus532 User Manual.*
2. *Priam InnerSpace High Performance Disk User's Manual* P/N 780075 (ours came in the same box as the disk).
3. *Priam InnerSpace Disk for the IBM AT Installation Instructions* P/N 780065 (ours came in the same box as the disk).
4. *IBM Guide to Operations: Personal Computer AT.*
5. *IBM AT Technical Reference Manual.*

5.10.1 Get a Priam Disk

We have extensive experience with both Model ID40-AT (formatted capacity 42 MBytes), and ID60-AT (formatted capacity 59.8 MBytes). Higher capacity disks are also available. These disks come with rails and cables especially for the AT, and are easy to install.

Contact Priam at 408/946-4600 for the dealer in your area. OEM discounts are available.

Priam's mailing address is 20 West Montague Expressway, San Jose, CA 95134.

5.10.2 Physically Install the Priam Disk

Follow the instructions in the *Priam InnerSpace Disk for the IBM AT Installation Instructions* P/N 780065. You can put the Priam disk in either the "first" (middle) or "second" (right, viewed from the front) physical position. Software works the same either way. The cabling, not the physical position of the disk, determines whether the disk is `c:` or `d:` for software.

The disk is shaped rather like a grand piano; slide it in with the curvy side toward the back of the PC.

Hints on Cabling: The disk connected to J4 will be `c:`, whether it resides in the "first" or "second" drive position. Similarly, the `d:` disk is connected to J3. The cables must have the red edge up (toward the ceiling). The red edge must also point to the left (viewed from the front of the PC) as they come out of the disk.

5.10.3 Run AT SETUP

Perform the following steps to run AT SETUP. The purpose of running SETUP now is to tell the PC about the new Priam disk.

1. Insert the IBM Diagnostics Diskette.
2. Type ctrl-alt-del to boot the diagnostics diskette.
3. The following menu appears:

```
SELECT AN OPTION

      0 - SYSTEM CHECKOUT
      1 - FORMAT DISKETTE
      2 - COPY DISKETTE
      3 - PREPARE SYSTEM FOR MOVING
      4 - SETUP
      9 - END DIAGNOSTICS

SELECT THE ACTION DESIRED
?
```

Choose option (4). This enters SETUP.

SETUP asks you to set and/or verify the date and time. Then it presents you with the following menu.

```
Your system may have other options
installed.  They are not required for
Setup and are not displayed.

The following options have been set:

Diskette Drive A -
Diskette Drive B -
Fixed Disk Drive C -
Fixed Disk Drive D -
Base Memory Size -
Expansion Memory Size -
Primary display is:

Are these options correct (Y/N)
?
```

Some information may already be filled in, so your display will probably not look exactly like this. Note the type(s) associated with previously installed hard disks (`c:` and/or `d:`). The type is a number from 0 to 14. If you are installing the Priam disk as a replacement for another disk, and you now intend to put the old disk in another machine, you will need to specify this type correctly when you re-install the old disk. Then type "N".

4. SETUP now goes through each device in the list and asks you to verify or change the information. When it asks about the fixed disks, indicate that the information is not correct. When it asks you the number of fixed disks on your system; answer "1" or "2" as appropriate. Then it asks for the type(s) of the fixed disk(s). The type is a number from 0 to 14. Use the numbers below as appropriate:

Priam Disk	SETUP Type
ID40-AT (40 MBytes)	11
ID60-AT (60 MBytes)	12

If your AT has a disk already installed, its type number will also be displayed by default, probably next to

```
Fixed Disk Drive C.
```

5. When all the information is correct, SETUP resets the PC. SETUP will start up again if you leave the diagnostics diskette in the drive; in this case, type "9" to exit the diagnostics and remove the diagnostics diskette.

   If the power-on self-test fails and times out, there is a problem. Check cabling and jumpering.

5.10.4 Create DOS Partition

Determine if you want a DOS partition on your Priam disk. If so, follow the steps below. If you do not want a DOS partition on your Priam disk, and you want to dedicate the disk to UNIX, the Opus **opconfig** program will handle the formatting and partitioning later. Skip this section and go directly to Section 5.10.5.

1. Put the DOS system diskette in the diskette drive.
2. Type ctrl-alt-del to reboot DOS.
3. Type the following:

   ```
   A>FDISK
   ```

   Choose option (1) "Create DOS Partition". Create the DOS partition by specifying starting and ending cylinders. We usually create a 50-cylinder (3 MByte) DOS partition. Choose a size that is appropriate for your needs.

   Make the DOS partition active by choosing option (2) from the main FDISK menu.

4. Reboot DOS (ctrl-alt-del) if necessary, and issue the following commands to format the DOS partition and install the files from the DOS system diskette onto the disk.

```
A>FORMAT C:/S/V
A>C:
C>MKDIR \DOS
C>COPY A:*.* C:\DOS
```

5.10.5 Modify config.sys

This section describes how to modify the DOS configuration file **config.sys** for 40-Mbyte and 60-Mbyte Priam disks; how to change other DOS system files; and how to install **opdisk**.

5.10.5.1 Modify config.sys (Priam 40-Mbyte)

These instructions apply to the Priam ID40-AT only. If you have a Priam ID60-AT, follow the instructions in Section 5.10.5.2.

If your Priam disk is `c:`, boot DOS and modify the DOS file **\config.sys** using the DOS editor EDLIN as follows:

```
C>\DOS\EDLIN \CONFIG.SYS
i
device = \opdisk.exe c:981 5 0 65535 0 0 0 0 0 981 17
buffers = 32
files = 16
ctrl-C
e
```

If your Priam disk is `d:`, type `d:981` instead of `c:981` in the "`device =`" line above.

Skip Section 5.10.5.2 and proceed directly to Section 5.10.6.

5.10.5.2 Modify config.sys (Priam 60-Mbyte)

These instructions apply to the Priam ID60-AT only. If you have a Priam ID40-AT, follow the instructions in Section 5.10.5.1.

If your Priam disk is `c:`, boot DOS and modify the DOS file **\config.sys** using the DOS editor EDLIN as follows:

```
C>\DOS\EDLIN \CONFIG.SYS
i
device = \opdisk.exe c:981 7 0 65535 0 0 0 0 0 981 17 0
buffers = 32
files = 16
ctrl-C
e
```

If your Priam disk is `d:`, type `d:981` instead of `c:981` in the "`device =`" line above.

5.10.6 Modify DOS System Files

This is a good time to create or modify the DOS file **\autoexec.bat** to set your DOS path and specify any other actions you want to be performed when you boot DOS. For example, to create an **\autoexec.bat** file using EDLIN so your path includes the directories `\dos` and `\opus`, and UNIX is automatically booted, you would type the following:

```
C>\DOS\EDLIN \AUTOEXEC.BAT
i
path = c:\dos;c:\opus
chkdsk c:
cd \opus
unix
ctrl-C
e
```

Before UNIX is installed, the commands to go to the `\opus` directory and run **unix** will fail; ignore these error messages.

5.10.7 Install Opdisk

Insert the Opus diskette containing **opdisk** (the BOOT diskette) and type the following:

```
C>COPY A:OPDISK.EXE C:\
```

Remove the Opus diskette and reboot DOS (ctrl-alt-del).

5.10.8 Install UNIX

To install UNIX, follow the instructions in Chapter 3 or Appendix A of this manual.

If you have two hard disks installed, and both have the special DOS "hidden" files on them that boot DOS, DOS will boot from whichever one has an active DOS partition.

If you have two disks, you must add an entry to **opus.cfg** for the second disk. Be sure you have an entry that looks like this:

```
<dsk/1=d:>
```

6. DEVICES

The PC has direct access to all devices on the system. The Opus-supplied program **opmon**(1*), running on the PC under DOS, mediates between Opus5 and the PC to provide access to devices from Opus5. **Opmon** contains drivers for all devices in the system. The purpose of device drivers under Opus5 is to provide information for **opmon**. Only **opmon**'s drivers talk directly to the device controllers.

Opmon knows **UNIX** devices by the names specified in the file **opus.cfg**. These device names can be assigned and modified using the program **opconfig**(1*). In **opus.cfg**, the names are enclosed in angle brackets like this:

```
<Opus5_device=opmon_driver>
```

You can thus derive the syntax for any **opus.cfg** specification from the tables in this chapter, by examining the rows for "Opus5 device" "opmon drivers". Note that the **/dev/** portion of the pathname is not used in the **opus.cfg** Opus5 device specification.

The Opus5 kernel knows devices by the major and minor device numbers listed in the Opus5 directory **/dev**. These numbers can be accessed by listing the directory **/dev** using the command **ls -l**, as shown below.

```
crw-rw-rw- 1 root  sys    10,  0 Nov 16 16:58 lp
 |                         |   |
 |__ character device      |   |__  minor device number
                           |__  major device number
```

UNIX operating systems have two kinds of devices: block devices and character devices. Block devices are listed by **ls -l** with "b" as the first character in the line; character devices have "c", as shown above. Block devices do buffered I/O; character devices do not. Some devices, e.g., printers and terminals, are never accessed in block mode. Others, e.g.,

disks and diskettes, sometimes use block mode and sometimes character mode. Typically, block mode is used only when a device contains a file system. Character mode is sometimes referred to as "raw" mode — hence the "r" in /dev/rdsk.

Unusual devices, such as graphics boards or non-standard disks, have no standard **UNIX** kernel or **opmon** drivers. You must either write your own (Opus can assist you) or make special arrangements with Opus. Adding a new driver within the Opus 100PM system really amounts to adding *two* drivers: one on the kernel side and one within **opmon**.

There is only one kernel driver, DK, for all Opus5 disk devices. However, there are three **opmon** drivers for hard disks.

1. The `c:xt` driver uses DOS BIOS calls and is optimized for the IBM PC XT controller. This controller can be used with a number of different disks.
2. The `c:` driver uses "generic" DOS BIOS calls. Any BIOS-compatible disk and controller pair can use this driver.
3. The `fil0` driver uses DOS calls. This driver is for disks and controllers that cannot use BIOS calls.

The BIOS is code that resides in ROM on the PC system board or on controllers such as the XT disk controller; it provides a standard interface to the standard PC hard disks, or any other compatible hard disk and controller.

Both the PC XT hard disk driver and the BIOS driver support partition-based logical drives. However, disks that do not use the BIOS interface cannot use partition-based logical drives; they must use DOS file-based logical drives, and this involves some performance loss. Thus, non-BIOS-compatible disks can be used with the Opus 100PM, but are not recommended.

This chapter discusses the correspondence between device names in **/dev** and **opus.cfg**. For each major device number in **/dev** and for each **opmon** driver listed in **opus.cfg**, we present the kernel driver name, the kernel device name(s), and the

opmon driver(s) called by each kernel driver.

The following major device numbers are defined in Opus5:

Device Number	*Character*	*Block*
0	console	dk
1	-	-
2	sys	gn
3	mem	mt
4	dk	user
5	user	user
6	user	user
7	user	user
8	err	
9	as	
10	lp	
11	prf	
12	sxt	
13	dos	
14	vdi	
15	gn	
16	pcl	
17	pclc	
18	mt	
19	xtty	
20	ex	
21	xm	
22	xs	
23	cpt	

See section 7 of the *UNIX System V Administrator Reference Manual* for more information on some of the devices discussed here.

6.1 Console Driver

Opus5 driver:	`console`
Opus5 devices:	/dev/console, /dev/systty
major device number:	0 (character)
opmon drivers:	`console, doscon`

Two **opmon** console drivers are available: `console` and `doscon`. Both drive the system keyboard and monitor. As of Opus5 Release C3 and later, both support escape sequences for foreground and background colors of characters (see the DOS *Technical Reference Manual* on `ansi.sys`).

Only one console entry may appear in **opus.cfg**.

6.1.1 console

The `console` driver uses PC BIOS commands to control the keyboard and monitor. Its **terminfo**(4) definition is `TERM=opus-pc`.

In Opus5 Release C3 and later releases, the option `direct` is available. This option allows **opmon** direct access to controllers that can directly access the console without synchronization or contention problems. It is useful with the IBM monochrome display adapter (MDA), Hercules, Compaq, and other console controllers. Specify the option as follows:

```
<console=console(direct)>
```

See Section 5.9 for special **opus.cfg** options for use with Sessions.

6.1.2 doscon

The `doscon` driver uses DOS commands to control the keyboard and monitor. Its **terminfo**(4) definition is `TERM=opus-ansi`.

6.2 DK Driver

Opus5 driver:	`dk`
Opus5 devices:	/dev/dsk/0s0, ... /dev/rdsk/0s0, ... /dev/flpa, ... /dev/rflpa, ...
major device number:	0 (block), 4 (character)

opmon drivers:	`c:, d:, ...`
	`c:xt, d:xt, ...`
	`fil#  a:  b:`
	`a:bios  b:bios`

The **dk** driver controls two types of devices: block (buffered) disk devices and character (unbuffered) disk devices. Block **dk** devices are in the directory **/dev/dsk**; the files **/dev/flpa**, **/dev/flpb**, etc., are also block devices. Raw **dk** devices are in the directory **/dev/rdsk**; the files **/dev/rflpa**, **/dev/rflpb**, etc., are also raw devices.

Each block **dk** drive corresponds to a disk partition or DOS file and can contain one or more Opus5 file systems. The kernel treats these partitions or DOS files as disk drives, capable of being divided into *sections*.

Each such virtual drive controlled by the **dk** driver can contain up to 8 logical "sections", numbered 0 though 7. Opus5 file systems can be created on section boundaries with **mkfs**(1M). Section 7 is reserved for use by the system.

The major device number of block **dk** devices is always 0; the minor device numbers specify the virtual drive and the section within it. There can be up to 16 **dk** devices, numbered 0-15; numbers 13-14 are ordinarily reserved for **opmon** devices `<b:>` and `<a:>`, respectively; number 15 is reserved.

The naming conventions for block **dk** devices are as follows:

```
/dev/dsk/nnsn
           | |__ logical section number within
           |       DOS file (0-7)
           |
           |__ dk drive number (0-15)
```

For **dk** devices, the integer result of dividing the minor device number by 8 gives the drive number. For example, a minor device number of 12 for a **dk** device means drive 1 section 4 and is called **/dev/dsk/1s4**.

See **opus.cfg**(4*).

6.2.1 c:, d:, ... Devices

These **dk** drives signify disk partitions.

These devices take optional arguments, as shown below:

```
<dsk/0=c:(drive=#, part=#)>
```

where

dma — Transfer directly between the disk and the Opus board. (Performance is somewhat better using **nodma**.)

drive=# — the physical unit number. Defaults are:

IBM-type systems	
Drive #	Unit #
C	0
D	1
E	2
F	3

max1 — Don't transfer more than one 512-byte sector at a time. If a request is received for more than one sector, break it up.

max2 — Don't transfer more than two 512-byte sectors at a time. If a request is received

for more than two sectors, break it up.

maxany — **Opmon** by default no longer allows reading or writing a disk across a track boundary. This option allows disk reads and writes to cross both track and cylinder boundaries.

maxcyl — In reading or writing the disk, do not allow a transfer across a cylinder boundary. If a request would cross a cylinder boundary, break it up in such a way that it does not. TI Business Pro users should use this option.

nodma — Don't transfer directly between the disk and the Opus board. Instead, transfer using a local buffer in **opmon**. This is the default.

part=# — Overrides the specification of the partition type. See example below. Seldom useful.

seek — Allow processing during disk seek operations. This results in slightly better overall performance.

Examples:

`<c:>` partition type 10 on drive C:

`<c:10>` partition type 10 on drive C:

`<dsk/3=e:10(drive=1, part=12)>`
(IBM-type systems only) partition type 12 on drive D:, to be used for **/dev/dsk/3s0**. In effect, maps e:10 to d:12.

`<dsk/0=c:(maxcyl, dma)>`
No transfers across a cylinder boundary, and use DMA.

6.2.2 c:xt, d:xt, ... Devices

For the IBM PC XT only, Opus provides a hard disk driver that talks directly to the PC XT disk controller. This can enhance performance in some circumstances. Usage is the same as the `c:` and `d:` devices described above. Examples:

`<c:xt>` partition type 10 on drive c:

`<c:10.xt>` partition type 10 on drive c:

6.2.3 fil# Devices

`fil#` devices in **opus.cfg** are DOS files seen by the kernel as virtual disk drives. These devices serve the same functions as partitions for users without BIOS-compatible disks. Up to 16 such devices (`fil0`, `fil1`, take the following parameters:

```
<dsk/0=fil0(size=#,path=pathname)>
```

where

`size=#` the maximum size of the DOS file that will contain the Opus5 file system(s). The default is "no limit". During initialization, you must give the size of each Opus5 file system, so **opinit** can execute the command **mkfs**(1M); this keeps Opus5 from writing past the end of the file system. Specifying a maximum size here in **opus.cfg** is an extra precaution that forces the driver not to write past the end of the DOS file. This number is specified in 512-byte blocks.

`path=pathname` the DOS filename. By default, `fil0` uses DOS file \opus\opfs\opfil0, `fil1` uses \opus\opfs\opfil1, etc.

The diskette devices `a:` and `b:` are special `fil#` devices. They are **dk** devices that use DOS files on diskette drives a: and b:. They correspond to Opus5 devices **/dev/flpa**,

/dev/flpb, etc., and **/dev/rflpa**, **/dev/rflpb**, etc. These devices are used during system installation; the normal Opus5 tape commands such as **cpio**(1) and **tar**(1) also typically use these devices. These devices contain DOS files which are logical Opus5 disk devices.

Devices `a:` and `b:` default to:

```
<dsk/14=a:(path=a:\opfil)>
<dsk/13=b:(path=b:\opfil)>
```

6.2.4 a:bios, b:bios, ... Devices

This driver controls diskettes directly, rather than through a DOS file. These drives do I/O with PC BIOS calls. Devices controlled by these drives (`a:bios` and `b:bios`) are useful for reading diskettes which are not readable by either DOS or Opus5.

By default, these devices are associated with **/dev/flpA** and **/dev/flpB** (block devices), and **/dev/rflpA** and **/dev/rflpB** (raw devices).

6.3 SYS Driver

Opus5 driver:	`sys`
Opus5 device:	/dev/tty
major device number:	2 (character)
opmon device:	none

The Opus5 device **/dev/tty** is associated with major device number 2. It is a virtual device; it is a synonym for the controlling terminal for any given process.

6.4 MEM Driver

Opus5 driver:	`mem`
Opus5 devices:	/dev/null, /dev/mem, /dev/kmem
major device number:	3 (character)
opmon driver:	none

/dev/mem is an Opus5 special file that is an image of the core memory of the computer. See **mem**(7). **/dev/kmem** is the same as **mem** except that kernel virtual memory rather than physical memory is accessed. See **mem**(7). **/dev/null** is the null file. See **null**(7).

6.5 ERR Driver

Opus5 driver:	`err`
Opus5 device:	/dev/err
major device number:	8 (character)
opmon driver:	none

Minor device 0 of the **err** driver is the interface between an Opus5 process and the system's error-record collection routines. See **err**(7).

6.6 Serial Driver

Opus5 driver:	`as`
Opus5 devices:	/dev/tty0, /dev/tty1 ...
major device number:	9 (character)
opmon driver:	`com# comh# coma# ctm#`

The Opus5 TTY driver controls access to devices **/dev/tty0**, **/dev/tty1**, etc., which are associated with modems or terminals. The TTY driver calls the **opmon** `com#`, `comh#`, `coma#`, or `ctm#` driver.

com# devices are associated with the standard serial port(s) on the PC. There can be one or two **com#** devices.

comh# devices are associated with additional serial ports provided by the Hostess Multiuse Host Adapter by Control Systems, Inc (see Appendix E). There can be up to 16 **comh#** devices, named **comh1** through **comh16**.

coma# devices are associated with additional serial ports provided by AST board (see Appendix E). There can be up to 8 **coma#** devices, named **coma1** through **coma8**.

ctm# devices are associated with additional serial ports provided by the Computone ATvantage-M card, from Computone Systems, Inc (see Appendix E). There can be up to 24 **ctm#** devices, named **ctm0** through **ctm23**.

By default, the first **com#**, **comh#**, **coma#**, or **ctm#** device listed in **opus.cfg** corresponds to devices **/dev/tty0** and **/dev/tty128**, the second to **/dev/tty1** and **/dev/tty129**, and so on. This default can be overridden by the **tty#=** option. For example, to explicitly assign **/dev/tty2** to **comh1**, the **opus.cfg** entry is the following:

```
<tty3=comh1>
```

The following arguments apply to **com#**, **comh#**, and **coma#** devices (but not **ctm#** devices):

io=# The I/O base address. See table below for defaults.

Note that this argument is optional for **com#** and **coma#** devices, but required for **comh#** devices, since Opus cannot determine which I/O addresses might cause a conflict in your system. See Appendix E for more information.

int=# The interrupt level. The following table shows defaults for I/O base address and interrupt level for

each type of device:

	I/O base addr default	*int default*
com1, com2	searches 0x3F8,	4
	then 0x2F8	3
comh#	—	—
coma1-4	0x2a0	—
coma5-8	0x1a0	—

iq=# Input queue size (default = 512 bytes).

oq=# Output queue size (default = 512 bytes).

rts Use RTS to control incoming data.

dtr Use DTR to control incoming data.

cts Send data only if CTS = 1.

dcd Send data only if DCD = 1.

local Force CLOCAL.

modem Report Ring Indicator transitions.

The following arguments apply to **ctm#** devices:

address= The segment address of the Computone card.

port= The port number within the Computone card to be associated with this TTY device. By default, **ctm0** is associated with **/dev/tty0**, **ctm1** with **/dev/tty1**, and so on. The **port=** option allows you to override these defaults.

rts Use RTS to control incoming data.

dtr Use DTR to control incoming data.

cts Send data only if CTS = 1.

dcd Send data only if DCD = 1.

A **tty#** device created by adding 128 to the minor device number of a **tty#** device does not require Carrier Detect to be present before initiating communication across a serial line. This can be useful if you want the same modem to serve for

both incoming and outgoing calls, or if no modem is attached. See Section 5.3.

6.7 LP Driver

Opus5 driver:	`lp`
Opus5 devices:	/dev/lp, /dev/lp1 ...
major device number:	10 (character)
opmon driver:	`lpt#`

The Opus5 **lp** driver controls access to line printer devices. It calls the **opmon** driver `lpt#`.

The I/O base address can optionally be specified for **lp** devices, as follows:

```
<lp=lpt1(io=#)>
```

where "#" is the I/O base address. By default, Opus searches the following addresses:

0x3BC
0x378
0x278

If you are using one of these addresses, you do not need to specify a value for "io".

See Section 5.4 for a list of Opus enhancements to the standard **lp** driver.

6.8 PRF Driver

Opus5 driver:	prf
Opus5 device:	/dev/prf
major device number:	11 (character)
opmon driver:	none

The file **/dev/prf** is a pseudo-device with no associated hardware. It provides access to activity information in the operating system. See **prf**(7).

6.9 SXT Driver

Opus5 driver:	sxt
Opus5 devices:	/dev/sxt000, ... , /dev/sxt037
major device number:	12 (character)
opmon driver:	none

The **sxt** pseudo-devices are used by the shell layering facility **shl**(1). See **sxt**(7).

6.10 DOS Driver

Opus5 driver:	dos
Opus5 device:	/dev/dos
major device number:	13 (character)
opmon driver:	dos

The dos driver provides access to DOS system calls. It is used for the DOS interface from Opus5. See also opdos(1*).

6.11 Clock Driver

Opus5 driver:	clock
Opus5 device:	none
major device number:	none
opmon driver:	`clock`

Opmon provides Opus5 with a real-time clock. Regardless of the PC clock rate, the kernel expects **opmon** to simulate a 60-Hz clock. Therefore, in order to derive the 60-Hz clock rate, **opmon** needs to know the PC real-time clock rate. **Opmon** assumes that PCs (except for TI PCs) clock at 18.2 Hz; for TI PCs, **opmon** assumes 40 Hz. If your PC has a different clock rate (e.g., AT&T 6300s), you should modify **opus.cfg** via **opconfig** to adjust these defaults.

The defaults can be adjusted by specifying the clock frequency in Hz as **n**umerator and **d**enominator of a fraction.

```
<clock=clock(freq=n/d)>
```

For example, to change the default for the AT&T 6300, whose clock rate is 18.75 Hz, change `<clock=clock>` in **opus.cfg** to read

```
<clock=clock(freq=1875/100)>
```

If the clock rate on your PC is different from the defaults, and you do not change **opus.cfg**, Opus5 will run but keep incorrect time.

There is no device in the Opus5 directory **/dev** corresponding to the clock.

6.12 Tape Driver

Opus5 driver:	**mt**
Opus5 device:	/dev/mt/[0-3][lmh][**n**]
major device number:	18 (character) 3 (block)
opmon drivers:	**wangtek** or **cms** **catamount** **archive**

The **mt** driver controls cartridge (**wangtek**, **archive**) and 9-track (**catamount**) tape devices. The **wangtek** cartridge driver supports the Tecmar Q60H and Q60AT. The device can also be purchased directly from Wangtek. A synonym for **wangtek** is **cms**. For hardware reasons, this device does not support multi-volume cartridge activity.

The **archive** cartridge driver supports the Scorpion series of streaming tape drives from Archive Systems (Costa Mesa, CA). Of the two cartridge devices, only the **archive** drive supports multi-volume cartridge activity.

The **catamount** driver supports the PCT-9 interface card from Catamount (Simi Valley, CA). Compatible drives can be purchased from Catamount or other vendors.

Mt devices reside in the directory **/dev/mt**. Device names are assigned as follows:

- The first character is a digit from 0 to 3. This identifies the tape unit number (the minor device number). Up to 4 tape units are supported.
- The second character is either **l**, **m**, or **h**, for low, medium, or high density. The Opus default is "m".
- The third character is optional. If present, it is the letter **n**, which signifies "no rewind". Tape operations by default rewind the tape after completion; this option causes the tape NOT to rewind. You must then rewind the tape manually, or with the Opus-supplied **tape**(1*) command.

Minor device numbers for "no rewind" devices range from 4 to 7.

The **wangtek** driver can be specified with an optional I/O base address argument:

```
<mt/0=wangtek(io=#)>
```

where "#" is the I/O base address. If not specified, the I/O address defaults to 0x338.

The **archive** driver can be specified with the following arguments:

io= An I/O base address (optional). Default is 0x200.

dma= The number of the DMA channel to use. Defaults to 1.

The **catamount** 9-track driver can take the following arguments:

io=# The I/O base address. Defaults to 0x220.

tout=# Duration of timeout in seconds. Default is 300.

buffer=# Size in bytes of high-speed buffer. Default is 32768 (32 1024-byte blocks).

hispeed Use transport's high speed (100 ips) for all operations. Default is use low speed (25 ips) for read and write, high speed for all else.

lospeed Use transport's low speed for all operations. Default is use low speed for read and write, high speed for all else.

For most cases, the default **opus.cfg** entry should be adequate. If you have both a **wangtek** and **catamount** device, be sure to assign different minor device numbers to them.

6.13 Ethernet

Opus5 driver:	**ether** (consists of **xtty**, **ex**, **um**, and **xs**)
Opus5 devices:	/dev/ttyT0 - /dev/ttyT8 /dev/EXOS/*
major device numbers:	19-22
opmon drivers:	**xln**

The **ether** driver controls access to Ethernet devices. Currently, Excelan's EXOS 205 Ethernet Front-End Processor is supported.

The specification for Ethernet in **opus.cfg** can take the following arguments:

io=# The I/O base address. Default is 0x310.

int=# Interrupt level. Default is 3.

seg The segment to be used by the Ethernet card. Default is 0xd000.

Use the following syntax:

```
<ether=xln(io=#, int=#, seg=#)>
```

Although the Ethernet driver is included in both the standard **opmon** and kernel, the supporting hardware and software must be purchased separately from Excelan and Opus. See the *Opus Ethernet Release Notice* for further information.

6.14 PC Net BIOS

Opus5 drivers:	`pcl, pclc`
Opus5 devices:	/dev/net[0-3] /dev/ctrl
major device numbers:	16, 17
opmon driver:	`pcl`

The PC Net BIOS drivers control the PC Net. The **pcl** driver is the standard IBM PC Net channel. The **pclc** driver is the **pcl** control channel, used by the **pcldaemon**.

Although the PC Net drivers are included in both the standard **opmon** and kernel, the supporting hardware and software must be purchased separately from IBM and Opus Systems. See the *Opus PC Net Release Notice* for further information.

6.15 GN Driver

Opus5 driver:	`gn`
Opus5 devices:	/dev/udio /dev/vrt /dev/cgi
major device number:	15 (character) 2 (block)

opmon drivers:	`udio`
	`verticom`
	`gsscgi`

The **gn** ("generic") driver controls access to devices of various kinds which are not part of the standard **UNIX** device list.

/dev/udio The **udio** interface, which allows concurrent DOS and UNIX (see Chapter 5). Uses minor device number 208.

/dev/vrt Verticom M-16, M256. Uses minor device number 224.

/dev/cgi GSS*CGI interface Uses minor device number 240.

The **udio** driver takes the following optional arguments:

```
<udio=udio(int=#, io=#)>
```

where

`int=#` Interrupt level. Specifying a free software interrupt vector can improve **udio**'s character output speed.

`io=#` Hardware interrupt level. **Udio** performance can also be enhanced by running the Opus board with interrupts. See Chapter 8 for guidelines. If you change your interrupt level, you will need to change hardware switch settings.

The Verticom and GSS*CGI interfaces take no optional arguments.

These devices are specified in **opus.cfg** as follows:

```
<udio=udio>
<vrt=verticom>
<cgi=gsscgi>
```

Although the **gn** drivers are included in both the standard **opmon** and kernel, the supporting hardware and software for Verticom and CGI must be purchased separately from the appropriate vendor and Opus Systems. See Opus Release Notices for each product for further information.

6.16 VDI

Opus5 driver:	`vdi`
Opus5 device:	/dev/vdi
major device number:	14 (character)
opmon driver:	`gssvdi`

The **vdi** driver controls the Virtual Device Interface from IBM. This is a general graphics interface. It requires a graphics board such as the Enhanced Graphics Adapter from IBM. All supporting software is included in the standard release of Opus5. See Chapter 5 for further information.

The **vdi** driver takes no optional arguments.

It is specified in **opus.cfg** as follows:

```
<vdi=gssvdi>
```

You may specify either the VDI driver `gssvdi` OR the **gn** driver `gsscgi` but not both.

7. SYSTEM MESSAGES

This chapter describes system messages and error codes in Opus5: **opmon** messages, including Level 1 and Level 2 selftest messages; **opsash** messages; **opload** messages; and other Opus5 messages.

This chapter has the following sections:

7.1 A brief description of the Level 1 and Level 2 tests.

7.2 An alphabetized list of messages, their source, and meaning, excluding I/O messages.

7.3 I/O messages grouped according to device.

7.4 How to turn on **opmon** error reporting.

See the Opus manual pages **opmon**(1*), **opsash**(1*), and **opload**(1*) for more information. These pages are in Appendix F of this manual.

7.1 Level 1 and Level 2 Tests

When the Opus system is first invoked from DOS, a series of initialization and verification operations occur. You see two main series of tests, labeled `Level 1` and `Level 2`. The Opus monitor program **opmon** runs the Level 1 tests, while the Opus standalone shell **opsash** runs the Level 2 tests.

7.1.1 Level 1 Tests

When you type `\opus\unix` from the DOS environment, the file **unix.bat** is executed. This file sets the working directory to \opus, and then runs **opmon. Opmon** does the initial setup operations for the Opus board. It locates the board, loads and reads values from the Opus registers and proceeds to test the first 60K bytes of the Opus memory. For the duration of Level 1 testing, the processor on the Opus board is halted while **opmon** tests the Opus memory across the PC bus (i.e., utilizing the shared memory segment).

The messages on the PC console during the Level 1 tests are written by **opmon**. They indicate that a test is now executing by displaying the message `Level 1 Test: xx`. The `xx` is replaced with a hexadecimal value that indicates the number of the test being executed by **opmon**. Refer to Section 7.2 for a description of these values and their meanings.

If a Level 1 test detects a failure, the value that is displayed denotes the test in which the failure was detected. A status message is also output after the failure has been detected; this message provides more information about the nature of the failure. These types of messages are described in Section 7.2.

A brief list and description of the Level 1 tests follows below. The Level 1 testing is subdivided into five main categories:

1. Initial Status testing
2. Initial Memory Testing
3. Memory Data Pattern Testing
4. Memory Addressing Testing
5. Operational Status Testing

Each of the subtests within these five groups will now be described.

7.1.1.1 Level 1 Initial Status Testing

This group of subtests is defined by Level 1 test numbers 00-05. These subtests perform six quick checks of the Opus 100 registers, as follows:

subtest 00	checks the Opus 100 status following reset to ensure that it is correct (initially = 0).
subtest 01	checks the PC to Opus interrupt bit functionality.
subtest 02	checks for stuck or toggling bits in the status.
subtest 03	checks to be sure the run bit is negated after a reset command.

subtest 04 — checks to be sure that the run bit asserts after the run command is issued.

subtest 05 — references Opus 100 memory location zero to ensure that the board returns data and doesn't hang. This makes sure the Opus 100 memory system is alive.

7.1.1.2 Level 1 Initial Memory Testing

This group of tests checks basic Opus 100 memory data path functionality as well as the ability of the memory to perform read and write operations. The test group consists of four subtests:

subtest 10 — writes a pattern of 00 hex to the first byte of memory. It then reads the byte back to ensure it matches.

subtest 11 — writes a pattern of 55 hex to the first byte of memory. It then reads the byte back to ensure it matches.

subtest 12 — writes a pattern of AA hex to the first byte of memory. It then reads the byte back to ensure it matches.

subtest 13 — writes a pattern of 01 hex to the first byte of memory. It then reads the byte back to ensure it matches. This pattern also causes the parity bit for this byte to be written to a zero for the first time.

subtests 14-43 — repeat the same testing sequence that was performed on byte zero by subtests 10-13 for each byte in the first 60K bytes of Opus 100 memory that has an address with only one bit set. In other words, it tests byte 1, 2, 4, ... 2000, 4000, 8000.

7.1.1.3 Level 1 Memory Data Pattern Testing

The level 1 memory data pattern testing group of subtests builds on the testing performed by subtests 10-43. This group of tests exercises each address of the first 60K bytes of Opus 100 memory in the following manner:

subtest 50 — writes all memory bytes from 0000-EFFF with the data pattern of 0000. It then reads and checks each byte for the expected value. The test writes and reads words.

subtest 51 — writes all memory bytes from 0000-EFFF with the data pattern of FFFF. It then reads and checks each byte for the expected value. The test writes and reads words.

subtest 52 — writes all memory bytes from 0000-EFFF with the data pattern of 5555. It then reads and checks each byte for the expected value. The test writes and reads words.

subtest 53 — writes all memory bytes from 0000-EFFF with the data pattern of AAAA. It then reads and checks each byte for the expected value. The test writes and reads words.

subtest 54 — writes all memory bytes from 0000-EFFF with the data pattern of 0001. It then reads and checks each byte for the expected value. The test writes and reads words. This forces the low byte parity bit to be zero.

subtest 55 — writes all memory bytes from 0000-EFFF with the data pattern of 0100. It then reads and checks each byte for the expected value. The test writes and reads words. This forces the high byte parity bit to be zero.

7.1.1.4 Level 1 Memory Addressing Testing

This group of subtests performs an address integrity check of the Opus 100 memory system. It tests the first 60K bytes in

the following manner:

subtest 60 — writes a pattern to every word of memory that is equal to that word's address. It then reads each word to ensure total address integrity.

subtest 61 — writes a pattern to every word of memory that is equal to the complement of that word's address. It then reads each word to ensure total address integrity.

7.1.1.5 Level 1 Operational Status Testing

These next two subtests perform further status checking of the Opus 100 board. They check for the following things:

subtest 90 — resets the Opus 100 board and reads the status register. It checks to see that upper four bits of the status are zero.

subtest 91 — resets the Opus 100 board, loads a jump instruction into the first memory location (that points to itself for the jump target) and issues the RUN command, and then the GO command. It then checks the status to verify that the board has entered the correct state.

7.1.1.6 Level 1 Test Completion

After the preceding Level 1 testing has completed successfully, the value of the test number is set to FF, and the Opus 100 is reset. Memory is enabled, (i.e., a GO command is issued), and the Opus 100 memory is cleared to zero and the test number field now changes to say "PASS". This completes Level 1 testing.

7.1.2 Level 2 Tests

Once Level 1 testing has successfully completed and memory has been cleared to zero, **opmon** downloads **opsash** into the first 60K bytes of the Opus 100 memory. It then puts the Opus

100 into the RUN mode (see Chapter 8), which starts the board executing **opsash**. The messages at the console now inform you that the Opus5 standalone shell has been loaded and Level 2 testing has begun.

The field following the words `Level 2 Test:` normally begins to increment while the Level 2 testing is in progress. The incrementing continues while **opmon** waits for Level 2 termination status from **opsash**. The amount of memory on the Opus 100 board and the speed at which the Opus 100 processor is running determine the maximum value of the field. Upon successful completion of the Level 2 tests, the value changes to display `PASS`. If the Level 2 testing completes prematurely, `FAIL` is displayed. If **opmon** never receives a termination request from **opsash**, `TIMEOUT` is displayed.

If `FAIL` is displayed, **opsash** sends additional error information to **opmon**, which is then displayed. This additional error information describes the error detected by **opsash**. Refer to Section 7.2 for an explanation of the Level 2 error messages and their formats.

Opsash performs extended memory testing above the first 60K bytes of Opus 100 memory. This range of memory is not accessible by the PC (**opmon**) until **opsash** has started and set up the Opus 100 memory management chip and built the proper page tables in memory. This is why **opsash** must be loaded and started before testing may proceed. **Opsash** completes the testing of the remaining memory that has been installed on the Opus 100 board.

Once the Level 2 testing has successfully completed, **opmon** and **opsash** begin to communicate through a shared memory window in the Opus 100 memory. **Opsash** typically requests that **opmon** load the standalone **fsck** program into the Opus 100 memory. Then **opsash** transfers control to **fsck** which performs a consistency check of the UNIX file system. When the check completes, **opsash** regains control and loads UNIX into Opus 100 memory. UNIX then takes control, performing initialization, then executing processes.

7.2 Messages

?

Response to <alt-128>. Type RETURN for menu, <space> to cancel. See **opmon**(1*) for option explanations.

ABT TRAP ENCOUNTERED <STATUS=xx>

Any standalone program can produce a TRAP error. TRAP errors are always fatal. An ABT trap means an unexpected MMU abort occurred. The status bits report the Opus 100 hardware status; see Section 8.4.

bad blk on DK drive <dn>, section <sn>
bn = <bn> er = <er1>,<er2>

The file system residing on the indicated disk device has an inconsistency in its free block list, if the block number is out of the range of the file system.

<dn> is the Opus5 drive number (the minor device number divided by 8). See Section 6.2.

<sn> is the section number (minor device number modulo 8). See Section 6.2.

<bn> is the disk block number relative to the beginning of the section specified by <sn> on the drive specified by <dn>.

<er1> is the **er** value returned by **opmon**.

<er2> is the **ds** value returned by **opmon**.

bad count on DK drive <dn>, section <sn>
bn = <bn> er = <er1>,<er2>

The file system residing on the device is inconsistent with information maintained in memory. (See "bad blk on DK" above.)

Bad free count on DK drive <dn> section <sn>
bn = <bn> er = <er1>,<er2>

The file system residing on the disk device has an

inconsistent free block list. See "bad blk on DK".

BPT TRAP ENCOUNTERED <STATUS=xx>

Any standalone program can produce a TRAP error. TRAP errors are always fatal. A BPT trap means the Breakpoint instruction was executed. The status bits report the Opus 100 hardware status; see Section 8.4.

Configuration: 32.16 2MB console dos dsk/0 dsk/1 flpa flpb lp tty0 tty1

An inventory of your system, including system type, amount of memory, and devices, printed during boot by **opsash**. This list is derived from the **opus.cfg** file, and so it will differ from system to system.

Device error on <device name, device number>
bn = <bn> er = <er1>,<er2>

A device has encountered a physical problem in accessing a block.

If **opmon** error reporting is turned on, the Opus5 kernel usually reports **er** and **ds** messages in addition to the messages generated by **opmon**. Within the kernel message, <er1> is **er** and <er2> is **ds**. Both numbers are in hex.

Opus5 does not generate messages when an error is corrected after retry (er1=1) or corrected after error correction (er1=2). When messages are generated both by **opmon** and Opus5, the **opmon**-generated errors are output first, followed by a blank line, and then the Opus5 message.

Device error on DOS device <dn>
bn = <bn> er = <er1>,<er2>

The DOS driver has encountered a problem while communicating with **opmon**. <**dn**> is the minor device number for the DOS interface (usually 0). <**bn**> is 0. Note that errors in the execution of DOS calls are passed back for use by **opdos**(1*), and do not cause the kernel to print error reports.

dma abort: eia = <xxx>, pc = <yyy>
panic: memory

An MMU error has been detected during access of Opus 100 memory via the PC bus. **<xxx>** is the address of the data access and **<yyy>** is the Opus 100PM program counter location at the time of the error.

DVZ TRAP ENCOUNTERED <STATUS=xx>

Any standalone program can produce a TRAP error. TRAP errors are always fatal. A DVZ trap means an attempt was made to divide by zero. The status bits report the Opus 100 hardware status; see Section 8.4.

FLG TRAP ENCOUNTERED <STATUS=xx>

Any standalone program can produce a TRAP error. TRAP errors are always fatal. A FLG trap means an illegal flag was encountered. The status bits report the Opus 100 hardware status; see Section 8.4.

FPU TRAP ENCOUNTERED <STATUS=xx>

Any standalone program can produce a TRAP error. TRAP errors are always fatal. A FPU trap means a floating point exception was encountered. The status bits report the Opus 100 hardware status; see Section 8.4.

ILL TRAP ENCOUNTERED <STATUS=xx>

Any standalone program can produce a TRAP error. TRAP errors are always fatal. An ILL means an illegal operation was encountered. The status bits report the Opus 100 hardware status; see Section 8.4.

Insert diskette for drive A: and strike any key when ready.
Insert diskette for drive B: and strike any key when ready.

When two diskette drives are simulated on a single physical drive, these messages prompt the operator to insert the appropriate diskette when a change is

required.

```
** I/O req:  dix=xx cmd=xx st=xx er=xx ...
** I/O done: dix=xx cmd=xx st=xx er=xx ...
** I/O err:  dix=xx cmd=xx st=xx er=xx ...
```

I/O request and I/O done messages are output for specified device index numbers by trace logic, if enabled. I/O error message is ouput by error-reporting logic, enabled by <alt-128>. See Section 7.3 for more on I/O messages.

Level 1 Test: PASS

Level 1 test passed.

Level 1 Test: xx

Level 1 test running; xx is test phase. The phases are:

- 0x - status tests
  - 00 - initial (reset) status
  - 01 - interrupt (PC to Opus 100) status
  - 02 - stuck bits
  - 03 - run mode off
  - 04 - run mode on
  - 05 - touch memory while reset
- 1x - test Opus 100 memory byte 0
  - 10 - pattern 00
  - 11 - pattern 55
  - 12 - pattern AA
  - 13 - pattern 01
- 14-43 - test bytes 1-2-4-8...8000 as above
- 5x - test Opus 100 memory (constant data)
  - 50 - pattern 0000
  - 51 - pattern FFFF
  - 52 - pattern 5555
  - 53 - pattern AAAA
  - 54 - pattern 0001 (low byte parity)
  - 55 - pattern 0100 (high byte parity)
- 6x - test Opus 100 memory addressing
  - 60 - pattern = address
  - 61 - pattern = inverted address
- 9x - test status while executing

90 - initial status
91 - status while running
FF - reset Opus 100 and zero memory

See Section 7.1 for more information.

Level 1 Test: xx FAIL...
Level 1 test phase xx failed. See Section 7.1 for more information.

Level 1 Test: xx FAIL: DMA abort detected.
An MMU abort of a DMA cycle was detected during level 1 test.

Level 1 Test: xx FAIL: illegal segment
The system configuration file (**opus.cfg** by default) specifies a segment that cannot be used by Opus5.

Level 1 Test: xx FAIL: memory location nnnn; wrote nn, read nn, then nn.
Level 1 memory test failed; the results of two successive reads are reported.

Level 1 Test: xx FAIL: parity error detected.
A memory parity error was detected during Level 1 test.

Level 1 Test: xx FAIL: status bits nn always on; bits nn sometimes on.
Level 1 status test failed for the reasons indicated. The status bits mentioned here are PC status bits.

Level 1 Test: xx FAIL: status should be nn but was nn, then nn.
Level one status test failed; the results of two successive status reads are reported. The status bits mentioned here are PC status bits.

Level 2 Test: FAIL

Level 2 test terminated prematurely. See Section 7.1 for more information.

`Level 2 Test: PASS`
Level 2 test passed.

`Level 2 Test: TIMEOUT`
Level 2 test timed out. See Section 7.1 for more information.

`Level 2 Test: xx`
Level 2 test is running; xx is timer.

`MEMORY TEST: DMA: nnnnnn ERRORS`
The memory test found a DMA error; `nnnnnn` is the total number of DMA errors in hex.

```
MEMORY TEST: nKb PASSED nKb FAILED nnnnnn ERRORS
    BANK (1 MB)   -->          #0        #1        #2        #3
                          ++++++++  nnnnnnnn* ........ ........
ADDR-nnnn BANK n WRITTEN-nnnnnnnn READ-nnnnnnnn (BIT nn)
  or
ADDR-nnnn (BANK n) WRITTEN-nnnnnnnn READ1-nnnnnnnn, READ2-nnnnnnnn
 ...
```

`MEMORY TEST: PARITY: n ERRORS`
A memory error was encountered during a Level 2 Test. These messages are reported by **opsash**, which runs the Level 2 Test. "nKb" is a multiple of 128 Kbytes and represents the percentage of memory that passed or failed the memory test. "ERRORS" is the total accumulated number of errors. This number is in hex and can be very large even if only one chip is bad.

The next line displays a "map" of memory, showing which banks contain bad bits or parity errors. The numbers 0, 1, 2, etc. are 128-Kbyte bank numbers. Interpret the display as follows:

`....` If the bank does not exist

++++	If the bank is without errors
nnnn	If the bank contains bad bits; the bits on in the hex number "nnnn" indicate the bad bits
*	If the bank contains a parity error

Then, for each of the first 8 memory errors, the test prints an ADDR= line such as one of the above. ADDR is the memory address (in hex) where the error occurred. BANK is the memory bank; each memory bank contains 128 Kbytes. Banks are numbered in hex beginning with 0, so bank 0 is 0-127 Kbytes, bank 1 is 128-256 Kbytes, etc. WRITTEN is the hex value written to the address, and READ is the value read back. An error is noted when the values for WRITTEN and READ differ. Two reads are done; an error is also noted if the results of the two reads do not agree.

The first form of the message listed above (with the BIT field) occurs when there is only one single-bit error, and the test can pinpoint it. The second form appears when the results of the first read differ from the results of the second; then the results of both reads are printed.

The rest of the errors, after the first 8, are neither printed nor saved.

MEMORY TEST: PARITY: nnnnnn ERRORS

The memory test found a parity error; nnnnnn is the total number of parity errors in hex.

no space on DK drive <dn>, section <sn>
bn = <bn> er = <er1>,<er2>

The file system residing on the disk device has no free blocks left. You should delete files to make more space. See "bad blk on DK".

opmon: argument error

This message is issued when **opmon** can't parse the optional arguments to the device specifications in **opus.cfg**. It is usually followed by an `opmon: configuration error` message, which reports the incorrect line in **opus.cfg**.

`opmon: bad magic number <num> in boot file: <name>`

The boot file (**opsash** by default) is in **a.out** format. An **a.out** file header contains a "magic" number, which identifies it as a file of a certain type, e.g., executable in the 32000 processor. This message is generated if the boot file has the wrong magic number.

`opmon: boot program too large: <name>`

The boot program (**opsash** by default) is larger than 60K bytes.

`opmon: can't open boot file: <name>`

Opmon tries to open the boot file (**opsash** by default) using a DOS open call. This message is issued if the open fails.

`opmon: can't open configuration file <name>`

Opmon issues this message when it tries to do a DOS open call on the configuration file (**opus.cfg** by default) and the call fails.

`opmon: cgi: can't find <int cgi>`

This message means the **opmon** driver for CGI did not find the correct version of the GSS*CGI library in DOS, and thus the DOS-resident driver for CGI was not loaded. A Driver Kit user with an outdated CGI library might see this message.

`opmon: configuration error: <filename> line <num>`

The configuration file (default **opus.cfg**) contains an error on the given line. Check syntax, spelling.

opmon: coprocessor segment not found

The configuration file specified "[S=?]" to cause a search of PC memory space for the Opus 100; the segment was not found. Possible errors include: Opus 100 not installed; Opus 100 segment switches set for invalid location; segment conflict (e.g., Opus 100 segment set for segment already containing memory or another device); Opus 100 hardware failure.

opmon: DMA abort; pc=xxxx:yyyy
[mmmm:nnnn]

DMA access to Opus 100 was aborted by the MMU; "xxxx:yyyy" specifies the program counter address in two 4-digit hex numbers representing segment and offset. If the program counter is within **opmon**, a second address "mmmm:nnnn" is printed, which represents the address relative to the beginning of **opmon**.

opmon: device index table overflow

When standalone **fsck** or **unix** is run, it gets a copy of the **opmon** device table. This message is issued if there is not enough space for this table.

opmon: device not found: <name>

The **opmon** device name listed in the configuration file does not correspond to any name in **opmon**'s internal table of devices.

opmon: DOS exec (cmd) failed

When you pause out of UNIX to go to DOS, a DOS process (typically the DOS shell) is forked. This message is given when the fork fails — for example, if the DOS command cannot be found.

opmon: error reading boot file: <name>

Opmon gives this message if it encounters a problem reading or seeking in the boot file (normally **opsash**).

opmon: insufficient memory (catamount)

opmon: insufficient memory (com)
opmon: insufficient memory (flp)
opmon: insufficient memory (lpt)
opmon: insufficient memory (wangtek)

The **opmon** drivers in parentheses in these messages allocate data buffers in DOS memory at boot time. These messages appear if there is insufficient DOS memory for the specified driver.

For **catamount**, the default buffer size can be made smaller (thus possibly solving this problem) by using the **buffer=** option to the **opus.cfg** specification for this device. For **com** devices, the input (**iq**) and output (**oq**) queue sizes can also be adjusted in **opus.cfg** to use less DOS memory. See Chapter 6.

Data buffers sizes for **wangtek**, **flp**, and **lpt** devices are of a fixed size, and cannot be modified.

opmon: too many devices; max = <num>

Too many devices specified in configuration file. Limit is specified by Opus5 kernel.

opmon: vdi: can't find <int vdi>

This message means the **opmon** driver for VDI did not find the correct version of the VDI library in DOS, and thus the DOS-resident driver for VDI was not loaded. A Driver Kit user with an outdated VDI library might see this message.

opmon: xln: sequence (entry) <num> <num>

This message is issued if Ethernet debugging is turned on. It simply gives the sequence number assigned to the current message by UNIX. It does not necessarily indicate an error.

opmon: xln: not compiled with DEBUG flag

If you are developing Ethernet drivers within UNIX (Driver Kit users only), a DEBUG flag is available in the compiler to assist you. The debugging information takes up space, so it is usually not compiled in. This

message appears if you try to use the debugging options when the program has not been compiled with the DEBUG flag.

opmon: xln: message out of sequence <num> <num>
Each message transferred between UNIX and the Ethernet card is assigned a sequence number. If a message is received out of sequence, this indicates a failure in the Ethernet driver. Contact Opus.

opmon: xln: board error; code = <num>
A hard failure has occurred in the Ethernet card. Call Excelan or Opus with the code number.

Out of inodes on DK drive <dn>, section <sn>
bn = <bn> er = <er1>,<er2>
The file system residing on the disk device has no free inodes. Delete files to make more inodes. See "bad blk on DK".

opsash: <filename>: not found
Opsash could not find the bootfile. Check pathnames and spelling.

opsash: <filename>: could not load
Opsash found a file but could not load it. Check for permissions, file corruption.

opunix: can't open dos udio
Opunix tried to open the DOS **udio** device and failed.

opunix: dos udio ioctl failed
Opunix tried to do an *ioctl*(2) call on the DOS **udio** device, and failed.

opunix: unix udiodaemon not active
The **udiodaemon** is not active. Review the set-up described in Section 5.1.2.1.

opunix: unix udio not available

The **udio** interface is not available. Review the set-up described in Section 5.1.2.1.

opunix: unknown option -[n]

You have given **opunix** a command option it does not recognize.

Opus Systems
Opus5 PC Monitor 3.00

Initial logon message. 3.00 is revision; may vary.

Opus Systems
Opus5 TI PC Monitor 3.00

Initial logon message for (obsolete) TI PC version.

PANIC: opmon:
xln: findmsg:* no msg found ***

UNIX can't keep up with interrupts coming from the Ethernet card. Can signify a hardware failure.

PANIC: opmon:
iodone: cpq full

Requests for device access go first into a general queue for all devices and then into a specific queue for each device. This message signifies that the general queue is full.

PANIC: opmon
q full

Requests for device access go first into a general queue for all devices and then into a specific queue for each device. This message signifies that the device-specific queue is full.

parity error: pc = <xxx>
panic: memory

A parity error has been detected during access to Opus 100 memory via the internal bus. <xxx> is the Opus 100PM program counter location at the time of the error.

SVC TRAP ENCOUNTERED <STATUS=xx>

Any standalone program can produce a TRAP error. TRAP errors are always fatal. An SVC trap means a supervisor call was encountered. The status bits tell the state of the Opus 100 hardware; see Section 8.4.

TRC TRAP ENCOUNTERED <STATUS=xx>

Any standalone program can produce a TRAP error. TRAP errors are always fatal. A TRC trap means "trace on" call was encountered. The status bits tell the state of the Opus 100 hardware; see Section 8.4.

UND TRAP ENCOUNTERED <STATUS=xx>

Any standalone program can produce a TRAP error. TRAP errors are always fatal. A UND trap means an undefined opcode was encountered. The status bits tell the state of the Opus 100 hardware; see Section 8.4.

7.3 I/O Error Messages

This section describes I/O error messages generated when **opmon** error reporting is turned on. The subsection has three parts:

- 7.3.1 Error Message Format
- 7.3.2 How to Interpret the Dix (Device Index) Field
- 7.3.3 Device Error Messages (er=0x09) Grouped by Device

7.3.1 Error Message Format

Opmon gives device status with two codes, called **er** and **ds**. **er** is the generic error code, and **ds** is device-dependent status. **ds** is not always used, and when not used is 0.

If **opmon** error reporting is turned on, the contents of any I/O request block (IORB) with **er** not 0 are printed just before the IORB is marked as done. The message begins with "`I/O err:`" and the format is as follows:

```
dix=xx cmd=xx st=xx er=xx ds=xxxx ad=xxxx cnt=xxxx blk=xxxx
```

The meanings of each field are as follows.

dix = device index in hex. Assignments are as follows:

1 first `<device>` in **opmon** configuration file (**opus.cfg**)

2 next `<device>`

3 next `<device>`, and so on ...

The console and asynchronous devices (<com1>, <com2>, etc.) each take three **dix** entries: one each for input, output, and control. See Section 7.3.2 for an example of how to interpret the **dix**.

cmd = I/O command
- 0 = nop
- 1 = reset
- 2 = status request

3 = read
4 = write
5 = read; no wait
6 = I/O control

st = IORB status
0 = idle
1 = go
2 = busy
4 = done

er = error type (hex)

00: E_OK	no error
01: E_RT	ok after retry
02: E_CE	ok after correction
03: E_CMD	illegal command
04: E_DA	illegal device address (e.g., disk block number)
05: E_MA	illegal memory address (outside 0000-EFFF)
06: E_NA	device not available
07: E_RO	attempt to write read-only device
08: E_DE	data error
09: E_DEV	device error (details in device status word)
0A: E_DIX	illegal device index
0B: E_CAN	canceled by system request
0C: E_NI	no input available
0D: E_DMA	DMA abort (MMU abort during DMA)
0E: E_PAR	parity error during DMA
0F: E_CRIT	DOS "critical error" (int 24h) — (not currently used)
10: E_TO	device timeout
11: E_SYNC	IORB request with IORB status <> GO
12: E_BADB	bad block on disk (flagged block)

ds = device-dependent status

This field varies according to device; see below.

ad = memory address within Opus 100 segment.

Legal values are 0000-EFFF.

cnt = count (usually byte count) in hex

blk = block number (e.g., disk block number) in hex

7.3.2 How to Interpret the Dix

I/O messages can be reported by devices of the kinds listed below. The device names shown (such as `<dsk/0=fil0>` or `<flpa=a:>`) are in **opus.cfg** format. When a message is reported, the **dix** (device index) is given; this number maps to the position of the device in the file **opus.cfg**.

For example, if the **opus.cfg** looks like the sample below, the **dix**es are assigned as listed.

```
[s=?]
[i=7]
[tz=PST8PDT]
<clock=clock>
<console=console>
<dos=dos>
<dsk/0=c:>
<flpa=a:>
<flpb=b:>
<lp=lpt1>
<tty0=com1>
<tty1=com2>
<vdi=gssvdi>
<mt/0=wangtek>
<udio=udio>
<pcl=pcl>
<end>
```

Within this sample file, the **dix** (device index) numbers would be the following:

clock	1
console	2, 3, 4 (input, output, and control)
dos	5
c:	6
a:	7
b:	8
lpt1	9
com1	A, B, C (input, output, and control)
com2	D, E, F (input, output, and control)
gssvdi	10
wangtek	11
udio	12
pcl	13

7.3.3 Device Error (er=0x09) Messages Grouped by Device

This section lists messages generated for each kind of device listed in **opus.cfg**.

- 7.3.3.1 Asynchronous I/O
- 7.3.3.2 IBM PC Hard Disk (BIOS)
- 7.3.3.3 IBM PC XT Hard Disk (DIRECT)
- 7.3.3.4 IBM PC Diskette (BIOS Driver)
- 7.3.3.5 IBM PC Diskette (DOS Driver)
- 7.3.3.6 IBM PC Line Printer (Parallel Port)
- 7.3.3.7 DOS Driver
- 7.3.3.8 DOS-Based Virtual Drive Files
- 7.3.3.9 Tape Devices
- 7.3.3.10 PC Net Driver
- 7.3.3.11 Ethernet Driver
- 7.3.3.12 VDI Driver
- 7.3.3.13 CGI Driver
- 7.3.3.14 GN Driver

7.3.3.1 Asynchronous (Serial) I/O

Opus5 driver: AS
opmon devices: `<com#> <coma#> <comh#> <ctm#>`

These are messages from serial lines; serial lines are named devices `<com1>`, `<com2>`, etc., in **opus.cfg**. The messages are coded in bits and can be OR'd. They are output in hex.

0001	break detected
0002	parity error detected
0004	framing error detected
0008	input overrun (hardware)
0010	input overrun (software buffer)
0080	change in modem status detected
0100	DTR active (out) - data terminal ready
0200	RTS active (out) - request to send
1000	DSR active (in) - data set ready
2000	CTS active (in) - clear to send
4000	CD active (in) - carrier detect
8000	RI active (in) - ring indicator

Example:

```
dix=0D cmd=03 st=02 er=09 ds=0380 ad=0000 cnt=0000 blk=0000000D
```

This means that on the device having the device index number 0D (`<com1>` in our example), when a read (03) was attempted, with IORB status busy (02), a device error (09) occurred of type 0380 (DTR active, RTS active, change in modem status detected).

7.3.3.2 IBM PC Hard Disk (BIOS)

Opus5 driver: DK
opmon devices: <c:> ...

The codes issued by the IBM PC BIOS (basic input/output system) disk driver are listed here. The BIOS is code that resides in ROM on the PC system board or on controllers such as the XT disk controller. It provides a standard interface to standard PC hard disks or any other compatible hard disk and controller; this interface allows programs to access the disk without having to be concerned with its device address or operating characteristics.

This device is used only if you use partition-based Opus5 virtual drives.

Opus codes (hex):

DE bad ecc on disk read - not correctable
CE bad ecc on disk read - corrected
BD bad block

AT only (hex):

E0 status error/error reg=0
CC write fault on selected drive
AA drive not ready

XT only (hex):

FF sense operation failed

AT or XT (hex):

BB undefined error
80 attachment failed to respond (timeout)
40 seek operation failed
20 controller failed
09 data crosses 64K boundary
07 drive parameter activity failed
05 reset failed

04 requested block not found
02 address mark not found
01 bad command passed to disk I/O

Example:

```
dix=20 cmd=03 st=02 er=09 ds=0004 ad=0840 cnt=0200 blk=0000000D
```

This means that on the device having the device index number 20 (not listed in our example because not part of standard **opus.cfg**), when a read (03) was attempted, with IORB status busy (02), a device error (09) occurred of type 0004 (requested block not found).

7.3.3.3 IBM PC XT Hard Disk (DIRECT)

Opus5 driver: DK
opmon devices: `<c:xt>` ...

Opus provides a driver that talks directly to the PC XT disk controller, bypassing the BIOS.

This driver is used only if you use partition-based Opus5 virtual drives.

Opus codes (hex):

DE bad ecc on disk read - not correctable
CE bad ecc on disk read - corrected
BD bad block

XT only (hex):

00 no error detected
01 no index detected
02 seek did not complete
03 write fault
04 drive not ready
06 no track 0 signal
07 drive parameter activity failed
08 seek in progress

10 ID ecc error
12 no address mark
14 block not found
15 seek error

20 invalid command
21 illegal disk address

Example messages for the `<c:xt>` device are similar to those for the BIOS hard disk device, except that the **opus.cfg** file must contain an entry for `<c:xt>`.

7.3.3.4 IBM PC Diskette (BIOS driver)

Opus5 driver: DK
opmon devices: `<a:bios> <b:bios>`

The codes issued by the IBM PC BIOS diskette driver are listed here.

Opus codes (hex):

DE bad crc on disk read

AT or XT (hex):

80 attachment failed to respond (timeout)
40 seek operation failed
20 controller failed
09 data crosses 64K boundary
08 DMA overrun
07 drive parameter activity failed
04 requested block not found
03 write attempted on write-protected diskette
02 address mark not found
01 bad command passed to disk I/O

Example:

```
dix=0A cmd=04 st=02 er=09 ds=0080 ad=0D44 cnt=0014 blk=00000000
```

This means that on the device having the device index number 0A (`<a:bios>` in our example), when a write (04) was attempted, with IORB status busy (02), a device error (09) occurred of type 0080 (attachment failed to respond). This can happen if no floppy is loaded in the floppy drive.

7.3.3.5 IBM PC Diskette (DOS driver)

Opus5 driver: DK
opmon devices: **<a:dos> <b:dos>**

Opus provides a driver that talks to the PC floppy disk controller, via DOS interrupts 25h and 26h, instead of via the BIOS.

Opus codes (hex):

DE bad crc on disk read

er = e_dev (0x09):

80 attachment failed to respond (timeout)
40 seek operation failed
20 controller failed
10 bad CRC on read
08 DMA overrun
04 requested block not found
03 write attempted on write-protected diskette
02 address mark not found
00 other

Example messages for the **<a:dos>** device are similar to those for the BIOS floppy disk device, except that the **opus.cfg** file must contain an entry for **<a:dos>**.

7.3.3.6 IBM PC Line Printer (parallel port)

Opus5 driver: LP
opmon devices: `<lpt1>` ...

The driver for the PC line printer (listed as <lpt1> in the sample **opus.cfg**) sends the following messages (in hex):

80	busy
20	paper out
10	printer not selected (offline)
08	I/O error

Example:

```
dix=0C cmd=04 st=02 er=09 ds=0020 ad=0D44 cnt=0014 blk=00000000
```

This means that on the device having the device index number 0C (`<lpt1>` in our example), when a write (04) was attempted, with IORB status busy (02), a device error (09) occurred of type 0020 (paper out).

7.3.3.7 DOS Driver

Opus5 driver: DOS
opmon device: **<dos>**

The DOS driver enables Opus5 to make DOS system calls. It is listed as **<dos>** in **opus.cfg**.

The error messages for **<dos>** are the same as the messages for the DOS-based virtual drive files (see next section).

The **cmd** field in DOS I/O messages is not meaningful (always 00) for the **<dos>** device.

7.3.3.8 DOS-based Virtual Drive Files

Opus5 driver: DK
opmon devices: `<fil#> <a:> <b:>`

If partition-based logical disks are not available, Opus5 can reside in DOS files that are seen as physical disk drives by the kernel. These DOS-file-based virtual drives can reside on hard disk (`<fil0>`, `<fil1>`, etc.), or on floppy diskettes (`<a:>`, `<b:>`).

The messages for `<dos>` are the same as the messages for DOS-file-based virtual drives.

1 invalid function number
2 file not found
3 path not found
4 too many opened files
5 access denied
6 invalid handle
7 memory control blocks destroyed
8 insufficient memory
9 invalid memory address
A invalid environment
B invalid format
C invalid access code
D invalid data
F invalid drive
10 attempt to remove current directory
11 not same device
12 no more files
13 attempt to write on write-protected diskette
14 unknown unit
15 drive not ready
16 unknown command
17 data error (CRC)
18 bad request structure length
19 seek error
1A unknown media type
1B block not found
1C printer out of paper

1D write fault
1E read fault
1F general failure
20 sharing violation
21 lock violation
22 invalid disk change
23 FCB unavailable
24-4F reserved
50 file exists
51 reserved
52 cannot make
53 fail on INT 24

Example:

```
dix=05 cmd=00 st=02 er=09 ds=0009 ad=0D44 cnt=0014 blk=00000000
```

This means that on the device having the device index number 05 (**<dos>** in our example), with IORB status busy (02), a device error (09) occurred, of type 0009 (invalid memory address).

7.3.3.9 Tape Devices

Opus5 driver: MT
opmon devices: `<wangtek> <catamount> <archive>`

This driver provides an interface to 9-track or cartridge tape drives, through the Opus-supplied **tape**(1*) command.

All **tape** operations print out a hex number when they complete. This number is the status of the tape drive at the conclusion of the requested operation. Bits in this status word are as follows:

bit 0	Tape write-protected.
bit 1	Tape drive offline.
bit 2	Beginning of tape.
bit 3	End of tape.
bit 4	End of file.
bit 5	Error.
bit 6	Command not implemented.
bit 7	Command timed out.

Example:

```
dix=05 cmd=03 st=02 er=09 ds=0002 ad=0D44 cnt=0014 blk=00000000
```

This means that on the device having the device index number 11 (<wangtek> in our example), during an attempted read, with IORB status busy (02), a device error (09) occurred, of type 0002 (tape drive offline).

7.3.3.10 PC Net Driver

Opus5 Driver: pcl, pclc
opmon device: `<pcl>`

The PC Net Driver enables you to use the IBM PC Network.

The following hex codes can be returned:

00	Good return
01	Illegal buffer length for SEND DATAGRAM, SEND BROADCAST, ADAPTER STATUS, or SESSION STATUS
03	Invalid command code
05	Command timed out
06	Message incomplete because buffer length too small
08	Illegal local session number
09	Not enough space available in the adapter for the session
0A	Session closed
0B	Command canceled
0D	Duplicate name in local name table
0E	Name table full
0F	Command completed, name has active sessions and is now de-registered
11	Local session table full
12	Session open rejected because no LISTEN command outstanding on remote
13	Illegal name number
14	Cannot find name called or no answer
15	Name not found, or cannot specify "*" in column 1 of the name field, or you specified 00H.
16	Name in use on remote adapter
17	Name deleted
18	Session ended abnormally
19	Name conflict detected
1A	Incompatible remote device (unexpected protocol packet received)
21	Interface busy (you called the BIOS out of an

	interrupt handler)
22	Too many commands outstanding
23	Invalid number in NCB_LANA_ NUM field
24	Command completed while cancel occurring
26	Command not valid to cancel
4X	Unusual network condition (X can be any hex value)
50-FE	Adapter malfunction
FF	The command is still pending

Example:

```
dix=13 cmd=03 st=02 er=09 ds=0003 ad=0D44 cnt=0014 blk=00000000
```

This means that on the device having the device index number 13 (`<pcl>` in our example), during an attempted read, with IORB status busy (02), a device error (09) occurred, of type 0003 (invalid command).

7.3.3.11 Ethernet Driver

Opus5 driver: ether
opmon driver: `<xln>`

The Ethernet driver allows you to use the Excelan Ethernet interface.

The following error codes can be produced:

01 Board having trouble

04 Read/Write during netload had errors

7.3.3.12 VDI Driver

See IBM or GSS documentation.

7.3.3.13 CGI Driver

See IBM or GSS documentation.

7.3.3.14 GN Driver

Codes for these drivers are device-dependent; see the vendor documentation for the appropriate devices.

7.4 Turning On Opmon Error Reporting

To turn on **opmon** error reporting from Opus5, press <alt-128>. (Type the ALT key and, simultaneously, the numbers 1, 2, and 8 *on the numeric keypad.*) This causes a question mark to be printed. Type "e" after the question mark as shown below.

```
$ <alt-128>?e
```

Either "on" or "off" is printed, depending on the current state. To turn off error reporting, do the same operation you used to turn it on. The "e" is a toggle; that is, typing "e" again reverses the effect of the previous "e".

Error reporting can be turned on in **opus.cfg** by placing [e] in the file.

8. Opus 100PM HARDWARE THEORY OF OPERATION

This section provides an overview of the Opus 100PM hardware, the hardware configuration options, and the operation of the Opus 100 coprocessor from a systems software viewpoint.

8.1 Overview

The Opus 100 consists of the following logical modules:

CPU — The CPU consists of a 32016 (Opus 108) or 32032 (Opus 110) CPU, a 32082 MMU, a 32081 FPU and a 32201 TCU.

Memory — The memory consists of a 1M-byte array (256K RAMs) with parity and a memory access/refresh controller. The memory has provision for a second 1M-byte array to be piggybacked on top of the first array. A 3 Mbyte piggyback board is available for the Opus 110PM. Depending upon configuration, this provides a total of 1, 2, or 4 Mbytes.

Peripherals — There are no onboard peripherals. All peripherals are under the control of the PC's microprocessor and are located either on the PC motherboard or on other boards in the PC chassis or expansion chassis.

Opus 100 DMA — The PC can access onboard memory through the MMU, which maps 60K bytes of PC address space into 512-byte pages of onboard memory. These accesses are typically made directly by the PC's microprocessor, but can be made (transparently to the Opus 100) by the PC DMA logic.

PC Bus Logic — Communication with the PC is accomplished via a 512-byte host communications page of memory accessed by the PC via MMU, and via several special control/status addresses. The PC can reset or interrupt the Opus 100 via the control ports, and the Opus 100 can activate a PC interrupt line to request service.

PC DMA — PC DMA channels 0-3 access Opus 100 memory normally. The IBM PC AT 16-bit DMA channels 4-7 do not transfer to Opus 100 memory.

8.2 Hardware Configuration Options

The Opus 108PM (32016 CPU) has the following configuration options. Note that the wait state jumper differs depending on hardware revision.

Opus 108 Option	*Implemented at Board Location*	
	Rev A0-C3	Rev D and up
Software option	D25 (sw 1)	D25 (sw 1)
Interrupt level	D25 (sw 2-4)	D25 (sw 2-4)
Segment number	D25 (sw 5-8)	D25 (sw 5-8)
Wait states	B16	B25

The Opus 110PM (32032 CPU) has the following configuration options:

Opus 110 Option	*Implemented at Board Location*
Software option	D2 (sw 1)
Interrupt level	D2 (sw 2-6)
Segment number	D2 (sw 7-10)
Wait states	H3 (JP2)

The 8-switch DIP switchpack at location D25 (Opus 108) or D2 (Opus 110) is used to implement three options: software option, interrupt level, and segment number.

Default settings for the switches at location D25 are shown in Figure 8-1.

Location D25 (Opus 108)

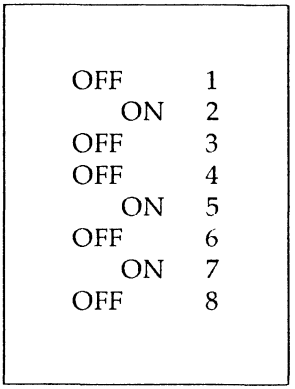


Figure 8-1. D25 Switch Settings

Default settings for the switches at location D2 are shown in Figure 8-2.

Location D2 (Opus 110)

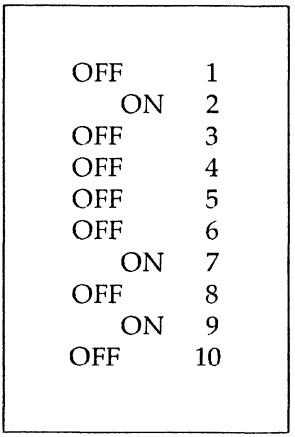


Figure 8-2. D2 Switch Settings

These default settings select:

- Software option off
- Interrupt level 7
- Segment number A

To help in changing any of these settings, the following sections provide technical details of each hardware configuration option.

8.2.1 Software Option

Switch 1 is readable by the Opus 100 processor and has no other hardware function. It should be left OPEN (OFF) unless required to be ON by software.

8.2.2 Interrupt Request Level

On the IBM PC bus, eight interrupt request levels are available:

Level		*Standard PC Function*	
(high)	0	Timer	
	1	Keyboard	
	2	Unused by the PC	(Opus option; 9 on AT)
	3	Communications	(Opus option)
	4	Communications	
	5	Hard disk	(XT)
	6	Floppy disk	
(low)	7	Printer	**(Opus default)**

The Opus 100 uses interrupt request level 7 by default. Levels 2 and 3 may also be used on the Opus 108. Levels 6, 5, 3, and 2 may also be used on the Opus 110.

On the Opus 108, selection of interrupt request level is accomplished by switches 2-4 of the DIP switchpack at location D25. On the Opus 110, selection of the interrupt request level is accomplished by switches 2-6 of the DIP

switchpack at location D2.

Opus 108 Switch Setting			*Selects Interrupt*
2	*3*	*4*	*Level*
ON	**OFF**	**OFF**	**7 (Opus default)**
OFF	ON	OFF	3
OFF	OFF	ON	2 (9 on AT)

Opus 110 Switch Setting					*Selects Interrupt*
2	*3*	*4*	*5*	*6*	*Level*
ON	**OFF**	**OFF**	**OFF**	**OFF**	**7 (Opus default)**
OFF	ON	OFF	OFF	OFF	6
OFF	OFF	ON	OFF	OFF	4
OFF	OFF	OFF	ON	OFF	3
OFF	OFF	OFF	OFF	ON	2 (9 on AT)

NOTE: The interrupt level specified in the Opus5 configuration file must correspond to the Opus 100 switch setting. Opus5 software is written to allow the Opus 100 to share interrupt level 7 with the PC printer controllers. If your application requires that you use an interrupt request level other than level 7, change the switches according to the table above and notify Opus5 via **opconfig** when you configure the system.

8.2.3 Segment Number

The IBM PC address space ranges from hex 00000 to FFFFF (1M byte). The ordinary PC memory segment usage is as follows:

Hex Address Range	*Standard PC Function*
00000-3FFFF	256K-byte memory
00000-7FFFF	512K-byte memory
00000-9FFFF	640K-byte memory
A0000-BFFFF	EGA modes 0-10
B0000-B7FFF	monochrome adapter, EGA monochrome emulation
B8000-BFFFF	CGA, PGA, EGA color emulation
C0000-C3FFF	EGA BIOS
C8000-C9FFF	XT disk BIOS
CC000-CDFFF	PC-NET BIOS
E0000-EFFFF	AT BIOS expansion
F0000-FFFFF	System BIOS

The shared memory segment for communication between the Opus 100 and the PC occupies 64K bytes and can be placed on any 64K boundary in this address space. Selection of 64K boundary is accomplished by switches 5-8 (7-10 on Opus 110) of the DIP switchpack at location 25D (D2 on Opus 110).

Opus 108 Switch Setting				*Selects Hex*
5	**6**	**7**	**8**	*Address Range*
ON	ON	ON	ON	00000-0FFFF
OFF	ON	ON	ON	10000-1FFFF
ON	OFF	ON	ON	20000-2FFFF
OFF	OFF	ON	ON	30000-3FFFF
ON	ON	OFF	ON	40000-4FFFF
OFF	ON	OFF	ON	50000-5FFFF
ON	OFF	OFF	ON	60000-6FFFF
OFF	OFF	OFF	ON	70000-7FFFF
ON	ON	ON	OFF	80000-8FFFF
OFF	ON	ON	OFF	90000-9FFFF
ON	**OFF**	**ON**	**OFF**	**A0000-AFFFF (Opus default)**
OFF	OFF	ON	OFF	B0000-BFFFF
ON	ON	OFF	OFF	C0000-CFFFF
OFF	ON	OFF	OFF	D0000-DFFFF
ON	OFF	OFF	OFF	E0000-EFFFF
OFF	OFF	OFF	OFF	F0000-FFFFF

Switches 7-10 (rather than 5-8) control the segment number on the Opus 110; otherwise, this table is exactly the same as for

the Opus 108.

NOTE: The segment number specified in the Opus5 software configuration file must correspond to the Opus 100 switch setting. Opus5 software is written to allow the Opus 100 to use any available segment number (normally 8, 9, A, D, or E). If your specific application requires that you choose a segment number other than A, change the switches according to the table above, and notify Opus5 via **opconfig** when you set up your system.

8.2.4 Number of Wait States

To optimize performance for a given combination of microprocessor speed (8 or 10 MHz) and RAM speed, the Opus 100 is jumpered to select some number of wait states. The Opus 100 can use 0, 1, 2, or 3 wait states, but generally uses only 0 or 1. The standard settings for the Opus 108 (Revs A0 through C3) are shown in Figure 8-3 below.

Opus 108 Rev A0 - C3 (Location B16)

Figure 8-3. Wait States, Opus 108

The standard settings for the Opus 108 Rev D and up, and all Opus 110s are shown in Figure 8-4.

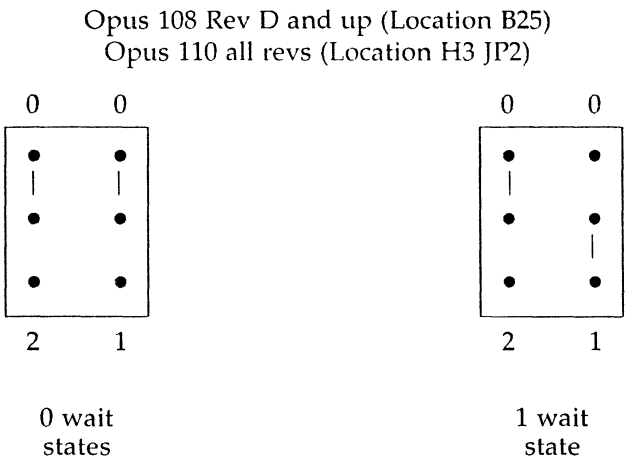


Figure 8-4. Wait States, Opus 110

8.3 Operation

See Section 8.5 for more information on the "commands" mentioned in this section.

Addresses — The Opus 100 can be located on any 64K boundary in the PC address space. This is accomplished by means of four address option switches, as described in Chapter 2 (Hardware Installation). The examples in this section assume that the switches are set for A0000:0, with control/status addresses starting at A000:F000.

Reset — The Opus 100 is initially held reset after power-up. The Opus 100 can also be reset by writing any value into shared memory location A000:F007. We call this issuing a RESET "command". Resetting the Opus 100 stops the Opus 100 CPU and puts all registers into a known state. The MMU goes into "physical mode"; that is, address translation is disabled.

Initialize — The initialize state is entered when the program in the PC issues a RESET command immediately followed by a GO command (writing any value into shared memory location A000:F005). This releases the system reset line, but forces a WAIT instruction to the CPU. In the initialize state, the PC

has access to the shared memory segment on the Opus 100 board, i.e., the first 60K bytes of onboard memory. The PC initializes the parity bits in the shared memory segment, tests the memory, and downloads selftest and initialization code for the Opus 100.

The MMU translates 24-bit virtual addresses. In the initialize state, special hardware on the Opus board constructs 24-bit addresses by putting zeroes in the most significant 8 bits of all addresses. This permits access to the first 60K of physical memory on the Opus 100 board. The last 4K bytes in this segment (normally A000:F000 - FFFF) are used for control functions as described in Section 8.5.

After memory is initialized, the program running in the PC (normally **opmon**) puts a bootstrap program (normally **opsash**) into the shared memory segment starting at location 0. This program cannot begin to run, however, until the go state is entered.

GO — The go state is entered when the program running in the PC (normally **opmon**) issues the "commands" RESET, RUN, and GO. The reset is released by the PC, and the program loaded into the first page of Opus physical memory (normally **opsash**) begins to run in the CPU. The RUN command causes ones to be put into the most significant 8 bits of all addresses coming into the MMU; it is now up to the program running in the CPU to re-vector these addresses to some page in Opus physical memory. **Opsash** performs a selftest, initializes the DMA map, and sets the IRQ interrupt to signal the program running in the PC that the Opus board is ready to be used. It also normally performs a system bootstrap.

All interaction between the Opus32 and the outside world is accomplished via the control/status addresses at A000:F000, and the host communications page.

Interrupts — By setting an interrupt, the Opus 100 can inform the PC that a request is pending in the host communication page. The PC performs the action requested, updates the block status, and interrupts the Opus 100 to indicate

completion. Typical interrupts might be for a timer interrupt, keyboard character, or disk transaction.

NMI — The non-maskable interrupt, triggered by the PC, serves as a break function.

8.4 Opus 100 I/O

Several special physical addresses can be read or written by the Opus 100 to monitor and control external events. These addresses are distinguished from memory addresses by the fact that they have bit 23 set to one (800000 hex). Note that these are physical addresses; the full 24-bit virtual address space is still available.

These I/O locations are:

Address	*Name*	*Function*
800000	WAIT	WAIT instruction
8000xx	BIT	Single bit output port (write only) (*Opus 110 only*) 800000: for Opus testing 800004: for Opus testing 800008: for Opus testing 80000C: not used 800010: not used 800014: not used 800018: not used 80001C: EWPE=1 force subsequent writes to be stored with bad parity (Opus testing purposes only)
810000	STAT	Status byte: (read only) (*Opus 110 only*) Bit 15: PERRO=0 parity error is in bytes 1 or 3 Bit 14: PERRE=0 parity error is in bytes 0 or 2 Bit 13: PERRH=0 parity error

Address	*Name*	*Function*
		is in bytes 2 or 3 Bit 12: PERRL=0 parity error is in bytes 0 or 1 Bit 11: NC undefined Bit 10: Opus testing purposes only Bit 9: FLT=0 parity error during MMU cycle Bit 8: HLDA=0 parity error during PC bus cycle
		(Opus 108 and Opus 110)
		Bit 7: INT=1 interrupt from PC Bit 6: IRQ=1 interrupt to PC active Bit 5: PAR=1 parity error Bit 4: DMA=1 DMA abort Bit 3: OPT=1 option switch open Bit 2: EIRQ=1 IRQ enabled Bit 1: CPU=0 this board Opus 110 CPU=1 this board Opus 108 Bit 0: always 0
810000	RSTE	Reset DMA and parity error latches
820000	ACK	Acknowledge (reset) interrupt from PC
830000	IRQ	Send interrupt request to PC

For functional details, see the following.

WAIT — Reading this location returns a wait instruction (0xB2). This location is normally used only by the hardware during the initialization state.

BIT — A single-bit output port. Only the least significant bit of the data being written is used by the output port. The remaining bits are not used by hardware. These bits are initialized to zero on power-up.

STAT — Reading this location returns eight bits of status on Opus 108, sixteen bits of status on Opus 110.

RSTE — This location is write-only; the actual bits written do not matter. Writing this location resets the DMA abort and

parity error latches. Both these errors cause NMI interrupts, and the NMI service routine should issue RSTE after the status byte has been read.

ACK — Writing this location resets the interrupt latch set by the PC to interrupt the Opus 100. It is normally done at the beginning of the interrupt service routine.

IRQ — This location is write-only; the actual bits written do not matter. Writing this location sets an interrupt request latch that interrupts the PC on level 7 (default). The state of this latch is available as status bit IRQ; the PC resets it when it accepts the interrupt.

8.5 PC I/O

Several locations in the PC address space are used for Opus 100 control/status functions. For locations that are to be written, the data written is irrelevant.

The control/status locations are:

Address	*Name*	*Function*
A000:F000	STAT	(*Opus 108 and Opus 110*) Bit 7: INT=1 interrupt to Opus 100 active Bit 6: IRQ=1 IRQ to PC active Bit 5: PAR=1 parity error Bit 4: DMA=1 DMA abort Bit 3: RUN* (diagnostic only) Bit 2: CWT* " Bit 1: TSO* " Bit 0: CTTL "
A000:F001	EIRQ	Enable IRQ interrupt to PC
A000:F002	ACK	Reset IRQ interrupt to PC
A000:F003	INT	Send attention interrupt to Opus 100

Address	*Name*	*Function*
A000:F004	NMI	Trigger Opus 100 non-maskable interrupt
A000:F005	GO	Release Opus 100 reset
A000:F006	RUN	Enable Opus 100 run mode
A000:F007	RST	Reset Opus 100

For functional details, see the following.

STAT — Reading this location returns 8 bits of Opus 100 status.

EIRQ — Writing this location enables the IRQ interrupt to the PC. It can be sensed in both the PC and Opus 100 status bytes, whether or not IRQ is enabled.

ACK — Writing this location resets the IRQ interrupt from the Opus 100. The reset should be performed at the beginning of the PC's interrupt service routine.

INT — Writing this location sets the attention interrupt for the Opus 100. The state of this interrupt is available in the status byte. It is reset by the Opus 100 when the interrupt is recognized.

NMI — Writing this location triggers an NMI interrupt in the Opus 100. The non-maskable interrupt can be used for special break conditions.

GO — Writing this location releases the Opus 100 reset. It will begin running at location 0 if RUN is given first, or enter the wait state otherwise. See also RUN.

RUN — Writing this location enables the Opus 100 for normal operation when the GO command is subsequently issued.

RST — Writing this location resets and disables the Opus 100. See also GO and RUN.

APPENDIX A: Opus5 SOFTWARE INSTALLATION USING DOS FILES

This appendix provides a step-by-step description of the procedures necessary to install the Opus5 software. This procedure differs from the one described in Chapter 3 because it uses DOS files as Opus5 logical disks. This procedure is recommended only if you cannot use partitions for Opus5 logical disks. You MUST use partitions if you need file systems larger than 28 Mbytes.

Not surprisingly, the installation procedure can be more or less complicated depending on system configuration requirements. Unusual configuration requirements include the following:

- more than one file system
- more than one hard disk drive
- a swap area larger than 2 Mbytes
- unusual segment number or interrupt level

The letter C: is used to describe the hard disk drive, according to standard IBM naming conventions. If your system uses different names — for example, Texas Instruments systems use the letter E: for the first hard disk — be sure to expect that letter instead.

After following the installation procedures in this appendix, you should perform the administrative set-up procedures appropriate to your system. The most common of these procedures are described in Chapter 4.

A.1 Installation Checklist

The following steps comprise installation using DOS files:

1. _____ Determine how many disk blocks you need to allocate for Opus5. (Section A.2)

2. _____ Back up DOS. (Section A.3)

3. _____ Run the program **opinit**(1*) to install basic software and run the configuration program **opconfig**(1*). (Section A.4)

4. _____ Run the interactive, menu-driven program **opload**(1*). (Section A.5)

The sections that follow describe these steps in detail.

A.2 Determine Number of Blocks for Opus5

You need a minimum of 10,000 512-byte blocks to store a small Opus5 operating system. This is about 5 Mbytes. You also need an additional minimum 2-Mbyte swap area, and at least 1 Mbyte for user files under Opus5. If you will be using your PC for DOS work, determine how much disk space you need for your DOS files, and compute your root file system size accordingly.

NOTE: Consider your choice carefully. Once the Opus5 logical disk has been created, its size cannot be changed without re-installing all of Opus5.

If you have a 10-Mbyte disk, you have about 20,000 total blocks.

To determine the number of free blocks on your system, run **chkdsk** under DOS, and divide the figure for "total bytes available" by 512. If you do not have enough space to create a large enough Opus5 logical disk, you should delete files accordingly.

A.3 Back Up DOS

CAUTION: BACKUP OF DOS FILES IS *HIGHLY RECOMMENDED.*

If you are installing system software from scratch (so you have no DOS files), or you have no DOS files you want to save,

you can skip this step.

1. Remove unnecessary files, e.g., redundant data files, scratch files, listing files and obsolete files. In addition, files available easily from another source, such as the DOS system distribution diskettes, could be removed.
2. When you are ready for backup, run **chkdsk** on the drive(s) to be backed up (see DOS manual) to determine how many diskettes you will need to hold the backup.
3. Using the DOS **format** command, format enough diskettes to hold the backup (using the size from **chkdsk**).
4. Backup the drive(s) with the DOS **backup** command (see DOS manual). For example, to backup all files on drive C: onto diskette drive A:

```
C>backup c: a:/s
```

This completes the procedure for backing up DOS.

A.4 Run opinit

The command "`opinit install`" handles most aspects of system installation and configuration. **Opinit** requires the following steps:

1. Insert the Opus5 BOOT diskette.
2. Type the following:

```
C>a:opinit install
```

This command does the following things:

1. It creates a directory called `\opus` under DOS on the (first) fixed disk, and copies the contents of the Opus BOOT diskette to that directory. As each file is copied, the filename is displayed. It then requests that you remove the BOOT diskette and

insert the KERNEL diskette. It copies more Opus5 files to DOS, including the Opus5 kernel.

2. It runs the program **opinst**, which begins running **opconfig**(1*), the system configuration program. See the next few subsections for in-depth information on **opconfig**.

   WARNING: **Opconfig** should be run by someone with a good understanding of the concepts of file system, and swap area, and also how **UNIX** operating systems treat devices. **Opconfig** is not a program to be run casually, since it alters the most basic system parameters and can write all over the disk(s).

3. After **opconfig** is finished, **opinit** requests that you load the ROOT diskette. It copies a small Opus5 file system into the swap area and boots this Opus5 operating system from the swap area, bringing Opus5 to "SINGLE USER MODE". It then requests that you load the "K" diskettes; these are the minimum files required to boot Opus5 from the hard disk and run the next installation command, **/opus/bin/opload**.

A.4.1 Running opconfig for System Initialization

Opconfig sets up system parameters and configuration information. It has the following six main modules:

(1) Standard System Initialization
(2) Opus5 Device Configuration (opus.cfg)
(3) DOS/Opus5 Partition Information
(4) Disk Bad Block Handling
(5) UNIX File System Layout
(6) Opus5 UNIX System Parameters

Opinit runs **opconfig** with option (1) "Standard System Initialization" in order to begin configuration of your DOS-file-based Opus5 system. Option (1) does the following:

- it requests your system type (e.g., TI, PC, XT, AT, etc.) and automatically adjusts device names and clock rate appropriately.
- it automatically invokes modified versions of options (2), (3), (4), and (5) as required.

The following subsections describe **opconfig** in more detail.

A.4.1.1 What Opconfig Does

Opus5 software uses information from **opconfig** during installation and booting.

Opconfig is intended to be self-explanatory for most installations. It is menu driven and has help information. See **opconfig**(1*) for more information.

You can type "exit" at any time to terminate **opconfig** and return to DOS.

A.4.1.2 Running Opconfig

Opconfig begins by asking you your system type (TI, XT, AT, etc.) so it can adjust clock rate and device names. It then verifies your time zone, and asks you to change it if necessary (help information is provided).

Some Opus5 system parameters are listed in the DOS file **opus.cfg**; others are part of the kernel. Part of **opconfig**'s job is to create and maintain the file **opus.cfg**; you should never have to modify that file directly in DOS (although you can). As part of standard system initialization, **opconfig** asks you to approve, and if necessary modify the standard mapping of kernel to **opmon** devices. See **opus.cfg**(4*) and **opconfig**(1*).

For DOS-file-based Opus5 systems, you MUST change **opus.cfg** to reflect the fact that you will use DOS files rather than partitions for Opus5 logical disks, in addition to any other changes you need to make to the file. You cannot use the default device list.

DOS files are specified in **opus.cfg** according to the format described in Section 6.2.3. If you choose to have one virtual drive for Opus5, delete the "`<dsk/0=c:>`" and "`<dsk/1=d:>`" devices and add the device "`<dsk/0=fil0>`".

If you need more than one DOS-file-based virtual drive, the second one should be specified as "`<dsk/1=fil1>`", etc., the third should be specified as "`<dsk/2=fil2>`", and so on. Use arguments to the `fil0` device names as appropriate (see Section 6.2.3). You must use arguments if the second DOS-file-based virtual drive is on a different physical disk. For example, if you have two physical disks, `c:` and `d:`, and each is to contain a DOS file-based logical disk, your **opus.cfg** entries might look like this:

```
<dsk/0=fil0(path=c:\opus\opfs\opfil)>
<dsk/1=fil1(path=d:\opus\opfs\opfil)>
```

If your DOS files are embedded several directory levels down, the DOS directories must already exist.

Combinations of DOS-file-based and partition devices are allowed, and in some cases, encouraged. For example, TI PCs are limited to one partition per physical disk, but often have two physical disks. TI PC users can create an Opus5 partition for one entire disk (say, F:), and then create a DOS-file-based Opus5 virtual disk on their second disk, to reside there along with DOS. Their **opus.cfg** entries for disk devices might look like the following:

```
<dsk/0=f:>
<dsk/1=fil0(size=10000,path=e:\opus\opfs\opfil0)>
```

See Section 6.2.3 for more information.

After **opus.cfg** has been modified to your satisfaction, **opconfig** reboots the system and restarts. It reviews your system type, time zone, and **opus.cfg** file, and then displays a table of the disk devices, so that you can check the DK devices. This table shows 0 for the number of blocks in your

logical disks; the purpose of this table is simply for you to verify that there are the correct number of them. The diskette devices, since they are controlled by the DK driver, are also shown in this list.

If you signify that this table shows the correct number of disks, **opconfig** then asks the number of 512-byte blocks in each DOS file that you will use for the Opus5 logical disk.

After you reply, **opconfig** gives you the option of subdividing your logical disk(s) into file systems. One file system per logical disk is the default; but additional file systems can be created on section boundaries, subdividing the logical disk. You can also change the default section boundaries.

Then **opconfig** enters module (4). This module ordinarily is used to spare bad blocks, but since you are using DOS files, DOS handles the sparing. Thus, the line for spare blocks is marked "NOT NEEDED".

You can use this module to change the size of the swap area if necessary. **Opconfig** first reports the number of blocks being used by the file system and swap area, as shown below.

```
The UNIX logical drive (dsk/0) is currently divided as follows:

  (1) Root file system:  16000 blocks (8 MB) 79%
  (2)        Swap area:  4000 blocks ( 2 MB) 20%
  (3) Spare block area:  202 blocks (0.4 MB) 1%
            Available:  0 blocks (0 MB) 0%

  Total partition size:  20202 blocks (10.4 MB)

Type "?" for more information, type "q" to go to the next partition,
  type the corresponding number [1-3] to change a value, or type
  "y" if these numbers are acceptable:
```

If you want to change the size of the swap area, choose option (2).

Type "y" when you are satisfied with the size of the swap area.

When **opconfig** is finished, **opinit** prompts you to insert the ROOT diskette, and, eventually, the K floppies. Opus5 is then booted in single user mode, and you are ready to run **opload** to load the rest of the Opus5 software.

A.4.2 If Installation is Interrupted

The first part of the "`opinit install`" procedure copies the BOOT and KERNEL floppies to disk. If you have interrupted the "`opinit install`" step after these floppies have been loaded and you wish to continue, you can issue the following command:

```
C>opinst
```

This begins the installation again, starting with the **opconfig** step. You can thus pick up the procedure without having to reload the BOOT and KERNEL floppies.

A.5 Run opload

Execute the following command to begin loading Opus5 from the Opus-supplied diskettes:

```
# opload
```

Opload(1*) is a menu-driven, self-documenting program. The first menu lists the groups of files contained in this release of Opus5. These group names (e.g., B, E, K, R) refer to Opus distribution diskettes or groups of diskettes. Descriptions of the contents and sizes of each group are in Appendix C. Use the **opload** menu to install files in this order:

- The required file groups B and E. (The K group is also required, but is already installed.)
- Other file groups or single files as you desire.

When you are finished, return to the Opus5 shell by typing "**q**" ("quit").

This completes the procedure for software installation of Opus5. Now perform system administration setup procedures as described in Chapter 4.

APPENDIX B: Opus5 vs DOS — HELPFUL HINTS

For users accustomed to the DOS environment, but new to Opus5, this section offers a few brief comparisons and offers a few hints regarding Opus5 operation.

Case Sensitivity — Opus5 is case-sensitive; DOS generally isn't. For example, under DOS, "DIR" and "dir" mean the same thing; however, under Opus5, the corresponding command "ls" would not be recognized if typed as "LS".

Time Zone — Internal Opus5 time is Greenwich Mean Time (GMT), not local time. Therefore, to cause Opus5 to keep local time, you must provide Opus5 with your time zone. See Section 4.3 for details on setting the time zone.

Sync(1) and **Fsck**(1M) — Two Opus5 commands are extremely important to successful system operation under Opus5. The two commands are **sync** and **fsck**.

> **sync** — Before shutting down under Opus5, always type **sync**. This causes Opus5 to update the superblock to reflect any changes that have occurred since the last sync was executed. (In multiuser mode, the system periodically executes a **sync** automatically). The **dos**(1*) command also executes a **sync** automatically.
>
> **fsck** — The Opus **unix** command automatically executes **fsck**, performing a file system check and verifying the integrity of the root file system. Before executing **mount**(1M) to mount another file system, you should run **fsck** on that file system.

Program Termination — Under DOS, programs are terminated with <ctrl+c> or <ctrl+Break>. Under Opus5, however, programs are terminated with an ASCII DEL character. This is generated from the PC keyboard with <ctrl+backspace>, NOT with the PC Del key. This applies generally to all Opus5 references to the DEL function. The <ctrl+Break> function may terminate Opus5 and return to DOS; use it with extreme

care.

Line Editing — The normal DOS line-editing function keys are not available under Opus5. The Opus5 backspace and line-cancel characters are initially # and @, but can be redefined by **stty**(1). **/opus/.profile** sets them to the backspace key and <ctrl+x>. The <shift+PrtSc> and <ctrl+NumLock> functions are available under both DOS and Opus5.

Command Analogs — Many DOS commands have analogous commands under Opus5. The following list shows many such commands. Operation and syntax may vary; refer to the appropriate manuals before using a new command for the first time.

DOS	*Opus5*
backup	cpio, tar
cd	cd
chdir	cd
chkdsk	fsck
command	sh
comp	diff
copy	cp, dd
date	date
debug	adb, sdb
del	rm
dir	ls, dir*
diskcopy	dd
edlin	ed, vi
erase	rm
find	grep
link	ld
mkdir	mkdir
mode	stty

***dir** under Opus5 is an Opus-supplied shell function defined in **/opus/.profile**.

DOS	*Opus5*
more	pg
path	PATH=
print	pr
prompt	PS1=, PS2=
recover	fsck
ren	mv
restore	cpio, tar
rmdir	rmdir
set	set, <variable>=
sort	sort
time	date
tree	find
type	cat

APPENDIX C: File Groups in the C3 Release

Opus5 Release C3 contains the following groups of files:

A — Accounting programs and related files

Group A contains system accounting files. These are most useful for large systems with many users. They are not appropriate for most Opus installations. We suggest that you not install group A.

B — Bin programs and related utilities

Group B contains the most important of the standard utilities. Group B should be installed.

C — C compiler and related programs and files

Group C contains the files necessary to compile and execute C programs. If you wish to do program development, you should install group C. (Group I and B should be installed as well.)

E — Etc maintenance programs and related files

Group E contains most of the system maintenance utilities. Group E should be installed.

F — F77 compiler and related programs and files

Group F contains the files necessary to compile and execute F77 programs. If you wish to do Fortran 77 program development, you should install group F. (Groups C and I should be installed as well.)

G — Games and related files

Group G contains educational and recreational programs. Of all the Opus5 programs, these are the least important; they are not necessary for the operation of Opus5 and they are not used in any standard application. If you need the disk space for more important files, do not install these programs.

I — Include files

Group I contains the include files necessary to compile C programs. If you want to do program development, you should install group I. (Group I should be installed when group C is installed.)

K — Key directories and files

Group K contains the key files necessary to have Opus5 run successfully. Group K must be installed.

L — Lint and related files

Group L contains programs and libraries related to **lint**(1), **lex**(1), the standalone C compiler and support libraries, as well as other C support programs such as **cb**(1), **ctrace**(1), and **cxref**(1). Group L is optional.

M — AT&T Reference Manual Pages

Group M contains the AT&T online manual. This information is the same as the hardcopy *User Reference Manual, Programmer Reference Manual,* and *Administrator Reference Manual.*

T — Terminfo files

Group T contains files necessary for running programs that use the terminfo capability (e.g. **vi**(1)). These files make up approximately half the files in the Opus5 release. It is likely that you need only a few. You should install only those files needed for the terminals you will be connecting to your system. You should not install Group T.

U — Utilities (major **/usr/bin** programs)

Group U contains Opus5 utilities of secondary importance. It is a fairly large group of files, only some of which may be useful for any given installation. Group U is optional.

V — Various utilities (minor **/usr/bin** programs)

Group V contains specific Opus5 utilities that are not needed at every installation, including SCCS and the files associated with the LP spooling system. It is a fairly large group of files, and should only be loaded by those requiring these programs. Group V is optional.

W — Historical utilities (minor **/usr/bin** programs)

Group W contains the least important Opus5 utilities. For the most part, this group contains files specific for the VAX environment that are not useful in the Opus5 environment. It is a fairly large group of files, only some of which may be useable for any given installation. Group W should not be loaded.

To help you plan the space allocation on your disk, here are the approximate sizes in 512-byte blocks of the various groups of Opus5 files. The entire release requires about 24500 blocks.

Group	*Size in Blocks*
K, B, E	4760
A	440
C	2050
F	1040
G	760
I	540
L	2020
M	3820
S	1100
T	950
U	2550
V	1750
W	1180

APPENDIX D: HINTS FOR PORTING F77 PROGRAMS

This appendix gives pointers on how to interface FORTRAN and C programs, and how to port FORTRAN 77 programs to run under Opus5. Some of this information is already contained in the standard AT&T documentation. The *UNIX System V Fortran-77 Reference Manual* from AT&T (publication number 308-278) is the primary reference. We also recommend that you have the *Series 32000 Instruction Set Reference Manual* from National Semiconductor Corporation (Customer Order Number NSP-INST-REF-M, NSC Publication Number 420010099-00 1B).

D.1 Procedure Names

These rules should be kept in mind.

1. Both C and FORTRAN put an underscore(_) in front of external procedure names.
2. FORTRAN also puts an additional underscore at the end of an external procedure name.

Examples:
The C function

```
foo() { ... }    becomes    _foo
```

The FORTRAN subroutine

```
SUBROUTINE FOO  becomes  _foo_
```

(Note: FORTRAN by default does not care about case. It can be forced to be case sensitive by using the f77 -U option.)

To call a C function from FORTRAN, use the statement below:

```
CALL ALPHA
```

The C function must be declared as follows:

```
alpha_()
{
        ...
}
```

This example assumes that the routine *alpha* does not return any value. If the C function returns a value, the function type must be declared explicitly. In FORTRAN the function declaration would be

```
INTEGER ALPHA, I
I = ALPHA(1)
```

The C function would then be

```
int alpha_(i)
int i;
{
        int intval;

        ...
        return (intval);
}
```

D.2 Data Representation

The following is a table of corresponding FORTRAN and C declarations.

C	*FORTRAN*
`int x;`	`INTEGER X`
`int x;`	`LOGICAL X`
`float x;`	`REAL X`
`double x;`	`DOUBLE PRECISION X or REAL*8 X`
`short x;`	`INTEGER*2 X`
`struct {float r,i;} x;`	`COMPLEX X`
`struct {double dr, di;} x;`	`DOUBLE COMPLEX X`
`char x[6];`	`CHARACTER*6 X`
`char x`	`CHARACTER*1 X`

Caution: Make certain that both FORTRAN and C understand the type of the functions to be the same.

Note that the first element of a C array always has subscript zero, but FORTRAN arrays begin at 1 by default. FORTRAN arrays are stored in column-major order, C arrays are stored in row- major order.

D.3 Return Values

A function of type integer, logical, real, or double precision declared as a C function returns the corresponding type. A complex or double complex function is equivalent to a C routine with an additional initial argument that points to the place where the return value is to be stored. Thus,

```
COMPLEX FUNCTION F( ... )
```

is equivalent to

```
f_( temp, ... )
struct { float r, i; } *temp;
...
```

A character-valued function is equivalent to a C-routine with two extra initial arguments: a data address and a length. Thus,

```
CHARACTER*15 FUNCTION G( ... )
```

is equivalent to

```
g_( result, length, ... );
char result[];
long length;
...
```

and can be invoked in C by

```
char chars[15];

g_( chars, 15, ...);
```

D.4 Arguments

All FORTRAN arguments are passed by address (CALL BY REFERENCE). In addition, for every argument that is of type character or that is a dummy procedure, an argument giving the length of the value is passed. (The string lengths are *int* quantities passed by value.) The order of arguments is then as follows:

1. Extra arguments for complex and character functions.
2. The address for each data object or function.
3. A long for each character or procedure argument.

Thus the following call in FORTRAN

```
CHARACTER*80 LINE
INTEGER I
DOUBLE PRECISION D
REAL R

CALL ALPHA ( I, D, LINE, R )
```

is equivalent to the following in C:

```
alpha_( i, d, line, r, length )
int *i;                    /* address of i */
double *d;                 /* address of d */
char *line;                /* address of line */
float *r;                  /* address of r */
int length;                /* length of line */
{
      ...
}
```

In the above example, note carefully that the *length* of *line* follows AFTER the regular arguments.

D.5 The 32000 Assembly Interface

Registers r0, r1, r2, r3, f0, f1, f2, f3, f4, f5, f6, f7 are treated as temporary registers. They are not saved across subroutine/function calls.

Register r0 is also used to return integer and pointer values from functions.

Register f0 is used to return single precision floating point values from functions.

The register pair f0,f1 is used to return double precision floating point values from functions.

Registers r4, r5, r6, r7 are used as register variables in C. They are also used for aliasing loop variables in FORTRAN. They must be saved and restored when used.

The stack pointer (sp) is used for passing arguments. The NS32000 "tos" address mode is often used to pass arguments.

The frame pointer (fp) has the following functions:

1. Subroutine linkage
2. Access to local variables
3. Access to arguments

The static base register is currently set to zero.

It is very instructive to examine the assembly language output from the C or FORTRAN compilers. This can be done by the -S option to the compilers. The result will be in the corresponding ".s" file.

D.6 Porting FORTRAN Programs

This section gives hints for porting FORTRAN 77 programs to run under Opus5.

The Opus implementation of F77 adheres closely to the ANSI Fortran 77 standard. It follows the description in the *UNIX System V Fortran-77 Reference Manual* (the primary reference). The following list describes common porting problems, and notes exceptions to the primary reference where applicable.

1. The command line option "-I2" does not work. Instead, use the "integer*2" convention within your program.
2. Opus F77 invokes the long jump assembler by default. If you want to use the short jump assembler, compile using the "-Ta/bin/as" command line option; for example,

```
f77 -Ta/bin/as myprog
```

3. Opus has added a command line option to F77 to create a cross-reference. The option is "-x". The cross-reference (program_name.x) is put into the current directory.
4. Note that the syntax to include a file is

```
include "stuff"
```

   This statement is replaced by the contents of the file *stuff*.

5. F77 ordinarily puts a newline at the end of literal strings to be printed. For example,

```
      WRITE (*,10)
10    FORMAT (ENTER YOUR NAME',)
```

outputs the following:

```
ENTER YOUR NAME
_
```

where "_" represents the current cursor position.

You can disable this by putting a dollar sign ($) at the end of the line to be printed, as follows:

```
      WRITE (*10)
10    FORMAT (ENTER YOUR NAME',$)
```

This outputs the following:

```
ENTER YOUR NAME_
```

6. Records are exactly 80 bytes long in the Opus implementation of FORTRAN 77. Some implementations allow 80-byte records with a newline at the end (i.e., 80 plus 1); Opus does not.

7. The Opus implementation adheres to the IEEE-754 floating point standard. This means that if you try to convert a double-precision number to any representation not large enough to hold the entire number (e.g., double to single precision, or double to integer), you will get a floating point exception. You will NOT get the same answer with less precision, as on a VAX. This is because the double-precision format differs between the VAX and the Opus Floating Point Unit. On a VAX, the first word of a double-precision number has the same format as a single-precision number: a sign bit, 8 bits of exponent, and 23 bits of mantissa. On the Opus FPU, the first word of a double-precision number has a sign bit, 11 bits of exponent, and 20 bits of mantissa.

8. The ANSI Standard does not specify what happens when a character expression is assigned to, say, an integer. F77 happens to treat all the following the same:

```
I='ABCD'
I='A'
I=ICHAR('A')
```

To achieve the intended (albeit non-portable because of byte ordering) result of the first assignment, an EQUIVALENCE must be used in f77.

9. The ANSI Standard suggests, but does not require, that Hollerith constants be treated as integers. F77 treats Hollerith constants as alternative notations for character strings. Therefore, the result of:

```
I=4Habcd
```

is the same as:

```
I='a'
```

or

```
I=ICHAR('a')
```

10. Numeric variables are aligned in common storage as follows:

TYPE	BYTE ALIGNMENT
integer	4
integer*2	2
real	4
double precision	4
complex	4
double complex	4

F77 gives a "common alignment" error for a variable *v* when *v* does not begin on the proper boundary. Usually this is because the variable immediately preceding *v* in the common list has upset the alignment; and usually the preceding variable is an integer*2.

To solve this problem, change the declaration of the common area so that integer*2 variables are listed last.

11. Note that the FORTRAN Standard does not specify where a file that has been explicitly **open**ed for sequential I/O is initially positioned. The Opus implementation attempts to position the file at the end, so a **write** appends to the file and a **read** results in an end-of-file indication. To position a file to its beginning, use a **rewind** statement. The preconnected units 0, 5, and 6 are positioned as they come from the program's parent process.

12. The maximum FORTRAN unit number in this implementation is 25.

    The maximum number of files that can be opened is 20. Remember that *stdin*, *stdout*, and *stderr* are 3 of the 20 files, so you can open 17 files in addition to these three.

APPENDIX E: ADDING SERIAL PORTS

Opus5 Release C3 supports three adapter boards for adding serial ports to the Opus 100PM system. These are:

1. The Hostess Multiuse Host Adapter by Control Systems, Inc (2855 Anthony Lane, Minneapolis, MN 55418). Up to two Hostess boards are supported, each of which supplies up to 8 serial ports, in addition to the standard serial ports.
2. The 4-port adapter by AST (2121 Alton Ave., Irvine, CA 92714). Up to two AST boards are supported.
3. The Computone ATvantage-M card from Computone Systems, Inc (1 Dunwoody Pk., Atlanta, GA 30338). Computone boards come in 4-port and 8-port models; up to 24 total Computone devices are supported.

This appendix, which is meant to supplement the vendor documentation, gives some general Opus-specific hints for using each of these boards.

E.1 Hostess Board

From **opmon**'s point of view, the additional serial ports on the Hostess board are `<comh#>` devices; from the kernel's point of view, they are TTY devices, controlled by the AS driver (see Section 6.7). `<comh#>` devices and `<com#>` devices share the same **opmon** driver.

You can use either 4 or 8 ports on the Hostess board. The connector at J3 (in conjunction with SW1 position 1 and SW2 position 8) determines how many ports you can use, according to Figure E-1 below.

Figure E-1. J3 Connector — # Ports

In order to use each Hostess board, you must modify **opus.cfg** to add **<comh#>** devices. You can modify **opus.cfg** either by using **opconfig**(1*), or by directly editing the file using **edlin** or WordStar in non-document mode. See Section 6.7 for format. You must explicitly specify the I/O address and interrupt level for **<comh#>** devices. You must also ensure that the I/O addresses you choose for the **<comh#>** devices do not conflict with anything else in your system. Since each **<comh#>** device takes up eight addresses, you need a total of 32 addresses for 4-port and 64 addresses for each 8-port Hostess board. Consult the IBM *Technical Reference Manual* or equivalent manual for your PC to determine which I/O addresses are in use.

The starting I/O address is selectable by switches on the Hostess board, as shown in Figure E-2.

Switch 1 — Set I/O Address (4-port Hostess)

8	7	6	5	4	3	2	1
on	on	on	off 200	off 100	off 080	off 040	OFF 020

Switch 1 — Set I/O Address (8-port Hostess)

8	7	6	5	4	3	2	1
on	on	on	off 200	off 100	off 080	off 040	OFF

Example: I/O Address = 0x280 (4-port)

8	7	6	5	4	3	2	1
on	on	on	off	on	off	on	on

Example: I/O Address = 0x280 (8-port)

8	7	6	5	4	3	2	1
on	on	on	off	on	off	on	off

Figure E-2. Hostess I/O Address Switches

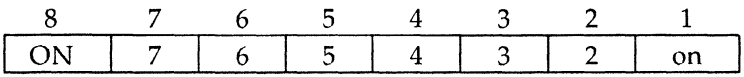
The table below summarizes these settings:

Switch	*Setting*
1	I/O 0x020 (4-port) off (8-port)
2	I/O 0x040
3	I/O 0x080
4	I/O 0x100
5	I/O 0x200
6	on
7	on
8	on

Each Hostess card must use a different base address and a different interrupt level. Each successive **<comh#>** device address is 8 greater than the previous one. For example, if the starting address is 0x280, the device **<comh1>** will have 0x280 as its "io_address".

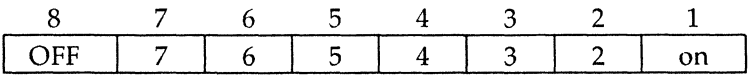
The interrupt level is also selectable by switches on the Hostess board, as shown in Figure E-3.

Switch 2 — Set Interrupt Level, 4-port Hostess



select one additional switch **on** for int 2-7

Switch 2 — Set Interrupt Level, 8-port Hostess



select one additional switch **on** for int 2-7

Figure E-3. Hostess Interrupt Level Switches

These settings are summarized in the table below:

Switch	*Setting*
1	on (mask enable)
2	on = int 2
3	on = int 3
4	on = int 4
5	on = int 5
6	on = int 6
7	on = int 7
8	on = 4 port off = 8 port

All `<comh#>` devices on the same card must share the same interrupt level. Thus, if the interrupt level for the first board is 2, and the interrupt level for the second board is 3, and the starting address for the first board is 0x280, and the starting address for the second board is 0x2C0, the entries in **opus.cfg** might look like this:

```
<tty0=comh1(io=0x280,int=2)>
<tty1=comh2(io=0x288,int=2)>
:
.
<tty7=comh8(io=0x2B8,int=2)>
<tty8=comh9(io=0x2C0,int=3)>
:
.
<tty15=comh16(io=0x2F8,int=3)>
```

`<com#>` and `<comh#>` devices are listed together in **opus.cfg**. You can explicitly associate particular UNIX TTYs with `com#` and `comH#` devices, using the `tty#=` convention in **opus.cfg**. For example, you can associate UNIX `tty4` with `comh2` by typing the following in **opus.cfg**:

```
<tty4=comh2(io=0x288,int=2)>
```

If you do not specify `tty#=`, then the order in which `com#` and `comh#` devices are listed in **opus.cfg** determines the assignment of kernel device names; that is, the first one listed — either a `<com#>` or a `<comh#>` — is associated with **/dev/tty0** (or **/dev/tty128**), the second is associated with **/dev/tty1** (or **/dev/tty129**), and so on.

On the kernel side, only two TTY devices (**/dev/tty0** and **/dev/tty1**) are created by default. You must use the **mknod** command to create others. For example, to create **/dev/tty3**, type

```
/etc/mknod /dev/tty3 c 9 3
```

In order to use these devices as login ports, you must create appropriate entries in the file **/etc/inittab**. See Section 4.10 for details.

A multiuser **UNIX** license is required if you intend to have more than a total of 2 login ports. Thus, for most applications, you need a multiuser license to use the Hostess board.

E.2 AST Board

From **opmon**'s point of view, the additional serial ports on the AST board are **<coma#>** devices; from the kernel's point of view, these are TTY devices, controlled by the AS driver (see Section 6.7). **<coma#>** devices and **<com#>** devices share the same **opmon** driver.

In order to use the AST board, you must modify **opus.cfg** to add **<coma#>** devices. You can modify **opus.cfg** either by using **opconfig**(1*), or by directly editing the file using **edlin** or WordStar in non-document mode. See the manual page for complete information on **opus.cfg**. You must explicitly specify the interrupt level for **<coma#>** devices in **opus.cfg**. If you are using two AST boards, each must have a different interrupt level. Please ensure that the interrupt level(s) you choose for the AST board(s) do not conflict with any other boards in your system.

Opus assumes that the first AST board uses I/O addresses starting at 0x2a0, and that the second AST board uses I/O addresses starting at 0x1a0, so you need not specify the I/O addresses in **opus.cfg**.

The interrupt level is selectable by switches on the AST board. All **<coma#>** devices on the same board share the same interrupt level. Thus, in *non-compatible* mode, if the interrupt

level for the first board is 2 and the interrupt level for the second board is 3, and the starting I/O addresses use the defaults, the entries in **opus.cfg** would look like this:

```
<tty0=coma1(,int=2)>
<tty1=coma2(,int=2)>
:
.
<tty3=coma4(,int=2)>
<tty4=coma5(,int=3)>
:
.
<tty7=coma8(,int=3)>
```

This assumes non-compatible mode. In *compatible* mode, the regular DOS device names are used for the first two ports. Also, in compatible mode, the interrupt levels for the first two ports must be unique. Thus, in compatible mode, **opus.cfg** entries might look like this:

```
<tty0=com1(,int=4)>
<tty1=com2(,int=3)>
<tty2=coma3(,int=2)>
<tty3=coma4(,int=2)>
:
.
```

`<com#>` and `<coma#>` devices are listed together in **opus.cfg**. You can explicitly associate particular UNIX TTYs with `com#` and `coma#` devices, using the `tty#=` convention in **opus.cfg**. For example, you can associate UNIX `tty4` with `coma2` by typing the following in **opus.cfg**:

```
<tty4=coma2(,int=2)>
```

If you do not specify `tty#=`, then the order in which `com#` and `coma#` devices are listed in **opus.cfg** determines the assignment of kernel device names; that is, the first one listed — either a `<com#>` or a `<coma#>` — is associated with **/dev/tty0** (or **/dev/tty128**), the second is associated with **/dev/tty1** (or **/dev/tty129**), and so on.

On the kernel side, only two TTY devices (**/dev/tty0** and **/dev/tty1**) are created by default. You must use the **mknod** command to create others. For example, to create **/dev/tty3**, type

```
/etc/mknod /dev/tty3 c 9 3
```

In order to use these devices as login ports, you must create appropriate entries in the file **/etc/inittab**. See Section 4.10 for details.

A multiuser **UNIX** license is required if you intend to have more than a total of 2 login ports. Thus, for most applications, you need a multiuser license to use the AST board.

E.3 Computone Board

Opus5 C3 and later releases support the Computone ATvantage-M card. These cards provide additional serial ports, analogous to `com1` and `com2`. Up to six 4-port or up to three 8-port cards, or any combination of 4- and 8-port cards, can be added to a PC with an Opus 100PM system installed, for a total possible 24 ports.

Earlier versions of the Computone board require 128K of PC address space (two contiguous segments); these are not recommended for use with the Opus board also installed. Later Computone cards require 32K (half a segment). Specifications for Computone ports in `opus.cfg` require you to specify the starting address in PC memory, using the format "`address=`*segment*". You must determine which segments in your system are available and set switches on the Computone card accordingly. This can be complicated; check your documentation from Computone. The segment you specify in `opus.cfg` must match the 16-bit address specified in the switch setting. For example, to specify Computone ports in `opus.cfg` starting at segment 8, type:

```
<tty0=ctm0(address=0x8000)>
```

The "**0x**" prefix is required. See Chapter 6 for additional arguments that apply to **ctm#** devices.

The Computone ports must be used as local terminals (**/dev/tty128**, **/dev/tty129**, etc.), as described in Section 5.3.

On the 8-port Computone card, ports are numbered from 0 to 7 (in **opus.cfg**, **ctm0** through **ctm7**). On recent versions of the 4-port card (with firmware optimized for Opus), ports are numbered from 0 to 3 (in **opus.cfg**, **ctm0** through **ctm3**); on earlier versions, they are numbered from 4 to 7.

If more than one 8-port card is installed, the **ctm** numbers are incremented sequentially, but the port numbering (0 through 7) is repeated for each card. For example, if the I/O address of the first card is 0x8000, and the I/O address of the second card is 0x8400, and TTY devices are mapped one-to-one with **ctm** devices, then the **opus.cfg** specifications would be as follows:

```
<tty0=ctm0(address=0x8000, port=0)>
<tty1=ctm1(address=0x8000, port=1)>
<tty2=ctm2(address=0x8000, port=2)>
<tty3=ctm3(address=0x8000, port=3)>
<tty4=ctm4(address=0x8000, port=4)>
<tty5=ctm5(address=0x8000, port=5)>
<tty6=ctm6(address=0x8000, port=6)>
<tty7=ctm7(address=0x8000, port=7)>
<tty8=ctm8(address=0x8400, port=0)>
<tty9=ctm9(address=0x8400, port=1)>
<tty10=ctm10(address=0x8400, port=2)>
<tty11=ctm11(address=0x8400, port=3)>
<tty12=ctm12(address=0x8400, port=4)>
<tty13=ctm13(address=0x8400, port=5)>
<tty14=ctm14(address=0x8400, port=6)>
<tty15=ctm15(address=0x8400, port=7)>
```

So if there is more than one card, there will be more than one port 0. But since each card's device name and address (**address=**) must be different, the address and name specifications are sufficient to differentiate the devices.

The list of **opus.cfg** lines shown above represents the default assignment of port numbers — in other words, ports are assumed to be in groups of 8. Thus, the "`port=`" specification is necessary only if multiple 4-port cards, or combinations of 4- and 8-port cards are used. If there are multiple 4-port cards, the port numbers only go from 0 to 3, and the numbering re-starts with each card. The I/O address is also different, of course, for each different card.

A multiuser **UNIX** license is required if you intend to have more than a total of 2 login ports. Thus, for most applications, you need a multiuser license to use the Computone board.

APPENDIX F: Opus5 REFERENCE MANUAL PAGES

The following are Opus5 reference manual pages in the format of the AT&T *UNIX System V Release 2.0 Reference Manual.*

NAME

dos - go to DOS or issue DOS command
dodos - make a DOS command a shell script

SYNOPSIS

/opus/bin/dos [**REMOTE**] [DOS command] [**BG**] [**WAIT**] [**QUIT**]
/opus/bin/dodos

DESCRIPTION

Dos with no arguments goes to DOS from within the Opus5 shell. To return to Opus5, type *exit* or *unix* from DOS. The command *resume* also works to return to UNIX, for compatibility with previous Opus releases.

Dos with a DOS command argument (e.g., **dos chkdsk**) executes that DOS command and returns to the Opus5 shell.

In C2 and later versions of Opus5, **opdos**(1*) requires that the pause and exit options be given from the console. The *dos* command is a shell script that uses both the pause and exit functions. Thus, by default, *dos* commands are prohibited if not issued at the console. The special command **dos** REMOTE overrides this; that is, it allows *dos* commands to be issued from a terminal other than the console (output of the command still goes to the console). Usage for this option is as follows:

```
# dos REMOTE [DOS command]
```

For example, to issue the DOS *chkdsk* command from a terminal other than the console, type the following:

```
# dos REMOTE chkdsk
```

The special command **dos BG** from the shell causes the system to go to DOS, while UNIX processes continue to run in background mode.

The special command **dos WAIT** from the shell causes the system to go to DOS, and suspends UNIX processes while the system is in DOS. The functionality of this command might seem redundant; the system by default suspends UNIX processes while in DOS. But you can change the default to **BG**; then **dos WAIT** overrides the new default.

The special command **dos QUIT** from the shell does a *sync*(1), and terminates both the current *opmon* and the current UNIX. The current UNIX session cannot be resumed after this; therefore, use this command with caution. This command is equivalent to typing "ALT-128" followed by "Q".

The command *dodos* is a shell script that executes **dos $@**. Thus, any one-line DOS command can be turned into a shell command by linking it to **dodos**. If the DOS command has the same name as a regular Opus5 command, e.g., **cd**, be careful to specify pathnames.

Using *exit* to return to UNIX requires no special preparation. However, if you use either the *unix* or *resume* commands to return to UNIX from DOS, you might need to modify your DOS path so you can access these commands from anywhere in the DOS directory hierarchy. The *unix* and *resume* commands reside in the DOS directory \opus. If you are placed in a directory other than \opus as a result of execution of your DOS command sequence, there are two ways you can access these commands:

1. Manually change directory to \opus before executing the *unix* or *resume*; or
2. Modify your DOS PATH (see DOS documentation) to include the \opus directory.

DOS batch files can be used with the *dos* command if they have *exit* as their last command.

DIAGNOSTICS

Same as *opdos*(1*).

SEE ALSO

opdos(1*)

NAME

ducp - DOS to Opus5 copy

SYNOPSIS

ducp [-a] DOS_file Opus5_file

DESCRIPTION

Ducp copies a DOS file to an Opus5 file. With the *-a* option, it performs an *opdos asciiread* operation on the Opus5 file, so that \r\n sequences are converted to ASCII newline characters for compatibility with Opus5. It recognizes ^Z as the end of a DOS file. With no option, it performs an *opdos read*, so no character conversion is done.

Ducp requires that both filenames be specified explicitly. To specify DOS pathnames from within Opus5, use two backslashes wherever you would ordinarily use one within DOS; this is necessary because the backslash \ is a special character for the shell. For example,

INCORRECT: `ducp \opus\file /usr/file`
CORRECT: `ducp \\opus\\file /usr/file`

In Opus5 releases C3 and later, *ducp* allows DOS-style wildcard substitutions. Note that these differ from UNIX-style wildcard conventions, in some important ways, notably that "*" can be used only at the *end* of a pattern.

DIAGNOSTICS

Same as *opdos*(1*).

SEE ALSO

udcp(1*), opdos(1*).

NAME

opconfig - configure Opus5 software

SYNOPSIS

C>opmon opsash; :opconfig [**-I** *filename*]
[**-O** *filename*] [**-P**] [**-M** *filename*]
[**-X** *filename*] [**-D**] [**-F** *numblks*]
[**-W** *numblks*]

DESCRIPTION

Opconfig is a program designed to automate the configuration and tuning of Opus5 systems. It can be used to examine and alter the following parameters:

— disk partitioning for Opus5 logical disks;

— the size and boundaries of your Opus5 file system(s);

— the size and starting address of your swap area if the defaults are not sufficient;

— the percentage of the root file system to be used for bad blocks, if the default (1%) is not sufficient;

— the segment of memory that will be the Opus segment;

— the system nodename for UUCP;

— the number of system buffers;

— the interrupt level you would like to use;

— your local time zone;

— your devices and how they will be mapped to Opus5 *opmon* and kernel drivers.

Opconfig is run automatically by the DOS batch files **opinit** and **opinst** during installation of C0 and later versions of Opus5 software.

Opconfig can read in a canned script and perform the appropriate actions. This feature can make the load procedure extremely simple. On the other hand,

using this feature makes error conditions difficult to determine, since the load either succeeds or not. See "Using Opconfig's Script Facility" below.

Most Opus5 system parameters cannot be changed while UNIX is running. Therefore, *opconfig* should generally be run from DOS, using the standalone shell, as shown in the SYNOPSIS above. However, you can use *opconfig* to examine current parameters while UNIX is running.

Opconfig is intended to be self-explanatory for most installations. It is menu driven and has help information. You can type "exit" at any time to terminate this program.

Unless you have unusual system requirements, you should respond in the affirmative to all *opconfig*'s queries, making as few changes as possible to the standard definitions.

Opus5 Release C2 and later *opconfig* takes the following command line options:

-I <*filename*> Read the script from the file *filename*.

-O <*filename*> Write a script to *filename*.

-P When used with the **-I** option, this displays each question and corresponding answer, and requires only a carriage return to continue.

-M <*filename*> The file *filename* contains a message that will be displayed before loading the root diskette. If *filename* doesn't exist, no message is displayed and there is no wait for a response.

-X <*filename*> The file *filename* contains the configuration file. The default is **:opus.cfg**.

-D Print out a dot as each command is read from the input file.

-F <*numblks*> Specifies the number of 512-byte blocks on the root diskette. The default is 700.

-W <*numblks*> Specifies the number of 512-byte blocks to be allocated to the swap area. The default is 4000.

Opconfig has the following six main modules:

(1) Standard System Initialization
(2) Opus5 Device Configuration (opus.cfg)
(3) DOS/Opus5 Partition Information
(4) Disk Bad Block Handling
(5) UNIX File System Layout
(6) Opus5 UNIX System Parameters

Each option is described below.

(1) Standard System Initialization

Chapter 3 and Appendix B describe this option in depth. Option (1) does the following:

— It requests your system type (e.g., TI, PC, XT, AT, etc.) and automatically adjusts device names and clock rate appropriately.

— It automatically invokes option (2) to modify the system configuration file **opus.cfg**.

— For installations using partition-based Opus5 logical disks, it leads you through modified versions of options (3), (4), and (5) as required. These options set up Opus5 partitions, file system boundaries, swap area, and spare area, and perform sparing of bad sectors.

(2) Opus5 Device Configuration (opus.cfg)

Option (2) allows you to examine and alter the configuration file **opus.cfg**(4*), which establishes the following Opus5 parameters:

— your local time zone (default PST8PDT) (*opconfig* provides online information to help you with format; see also Section 4.3);

— your interrupt level (default `[i=x]`);

— the Opus memory segment (default `[s=?]`);

— the attention key sequence (default `[a=128]`);

— your devices, and how they are mapped to Opus5 **opmon**(1*) and kernel drivers. Devices are listed in the format `<Opus5_device=opmon_driver>`, where "Opus5_device" is usually the name by which the device is known in the directory **/dev**; and "opmon_driver" is the name of the driver on the DOS side, which actually controls the device. See Chapter 6.

For **dsk** devices, "dsk/0" refers to a whole logical disk, capable of being subdivided on section boundaries into devices **/dev/dsk/0s0** - **/dev/dsk/0s7**.

See **opus.cfg**(4*).

The following shows a sample display for default values of Opus5 devices for a system using BIOS hard disk drivers, in the Pacific Time Zone.

```
Opus5 Device Configuration (modifying DOS file opus.cfg)

Your current configuration file includes:

PARAMETER     [tz=PST8PDT]     [i=x]          [s=?]
              [a=128]

SYSTEM        <clock=clock>    <dos=dos>      <console=console>

DISK          <dsk/0=c:>       <flpa=a:>      <flpb=b:>

TTY           <tty0=com1>      <tty1=com2>

LP            <lp=lpt1>        <lp1=lpt2>

SPECIAL       <vdi=vdi>

MISC

Is this configuration file okay [y,n]?
```

System parameters are listed first, in square brackets. Devices follow, surrounded by angle brackets; they are listed in functional groups. MISC devices in the display have no identifiable standard drivers.

To modify **opus.cfg**, answer "n" when asked if the configuration file is okay. You are then presented with the following menu:

```
You may modify this configuration as follows:

     (?)  More information
     (1)  Add an entry to the current configuration
     (2)  Delete an entry from the current configuration
     (3)  Modify an existing entry
     (4)  Construct opus.cfg from default values
     (5)  Reread opus.cfg from the DOS file system

     (q)  Return to top level

Choose an item from the list above or type "y" if okay:
```

Most choices in this list are self-explanatory. You might wish to construct **opus.cfg** from default values (option (4)) if you are installing from scratch, and you want to discard all the changes you have made in the current session and start over. Option (5) from this menu is like an "undo" command that rereads your configuration file from the DOS file system.

If you are using DOS files as Opus5 logical disks, you MUST modify the DSK device specification in **opus.cfg**, removing `<c:>` and replacing it with one or more `<fil#>` devices. See Chapter 6 and Appendix B.

Opus5 Release B0 **opus.cfg** files do not use the `<Opus5_device=opmon_driver>` convention for designating the assignment of kernel drivers to devices. If you attempt to use an old style **opus.cfg** file, *opconfig* automatically converts it to the new format.

(3) DOS/Opus5 Partition Information

Option (3) allows you to examine and alter the disk partitioning for Opus5 partition-based logical disks.

If you use DOS files for Opus5 logical disks, your Opus5 logical disks are DOS files within the DOS partition, so this option is not important for you.

Note that in Opus5 Release C0, the swap area and bad block area are by default in the same partition as the

root file system.

When *opconfig* is called in standalone mode (not as part of standard system initialization), option (3) begins by listing the current logical disk drives. A sample display is shown below. When option (3) is run as part of standard system initialization, this screen appears near the close of option (3) for final check.

```
Current available logical disk drives

Drive    Drive     Partition     Sector      Opmon
number   name        type         count      name

(0)      dsk/0       UNIX        16000      c:
(13)     flpb      DOS FILE                 b:\opfil
(14)     flpa      DOS FILE                 a:\opfil

To change any of the partition information for the above driv
type the drive number otherwise type "y" if okay:
```

The headings in this display mean the following:

Drive Number

The logical kernel drive number [0-15] for this drive. Diskette devices are listed here because they are controlled by the DK driver, which treats them as DOS file-based logical drives.

Drive Name

The logical kernel device name (**dsk/0**, **dsk/1**, etc.) for this drive. If the drive number is 0, the drive name is **dsk/0**; if the drive number is 1, the drive name is **dsk/1**, etc. Kernel names and numbers are assigned to physical drives by the assignment statements in **opus.cfg**, e.g., **<dsk/0=c:>**. Note that a disk driver name (e.g., **dsk/0**) corresponds to a logical disk that includes all the sections of

that disk (i.e., **/dev/dsk/0s0** through **/dev/dsk/0s7**).

Partition Type
: There are two partition types: "DOS FILE" and "UNIX". By default, the partition created during standard system initialization has type "UNIX". Previous releases allowed alternate partition types 11, 12, and 13; if you have these and do not want to re-partition, they will still work.

Sector Count
: The number of 512-byte sectors in this partition.

Opmon name
: Three **opmon** drivers can be associated with the UNIX **dk** driver: the first optimized for IBM PC XT controller (`x.xt`), the second for generic BIOS interfaces (x:), and the third for DOS interfaces (`fil#`). See Chapter 6. This column specifies which driver is associated with which DK device.

If any DK device is a DOS file, this is indicated in the "Partition type" column.

The standard system initialization by default sets up one UNIX type partition in addition to the DOS partition.

Only partition information can be changed using option (3); this means you can't apply option (3) to DOS files. If you determine that any partition needs to be changed, the following kind of display appears:

```
 Current Fixed Disk Logical Drive dsk/0

 Partition Status  Type Start  End Cyls Sectors
>         0      N UNIX     90  304  215   14609 (7.4 MB)
          1      N    0      0    0    0       0
          2      N    0      0    0    0       0
          3      A  DOS      0   89   90    6120 (3.1 MB)

Total Disk                   0  304  305   20729 (10.0 MB)

Modifying Logical Drive "0", Partition 0

        (0)  Kill this partition
        (1)  Modify other partition

        (2)  Type:       [DOS, UNIX, or type number]
        (3)  Start:      [Begin cylinder number]
        (4)  End:        [End cylinder number]
        (5)  Cyls:       [Total number of cylinders]
        (6)  Sectors:    [Total number of 512 byte blocks]

Choose an item from the list above [0-6] or type "y" if okay
```

The headings in the top portion refer to the following:

Partition — Opus5 typically occupies 1 of the 4 partitions available on the physical disk. By default, it is installed in the first partition, labelled 0. The numbers in this column are only for identifying and selecting rows in the display.

Status — Active or Non-active. To run Opus5, the DOS partition must be active, UNIX non-active. This setting cannot be modified.

Type — Usually either DOS or UNIX. Previous releases allowed partitions of types 11-13; this is still supported.

Start — The starting 512-byte block address of this partition.

End The ending 512-byte block address of this partition.

Cyls The number of cylinders in this partition. On IBM-type systems, there are about 30 cylinders in 1 Mbyte.

Sectors The number of 512-byte sectors in this partition.

Megabytes
The approximate number of megabytes in this partition.

The current values for type, starting sector, ending sector, number of cylinders, and total sectors are printed on the corresponding line in the second half of this display.

When specifying new partition boundaries, only start cylinder (3) and one of (4), (5), or (6) are required, because the program can figure out the rest.

If you wish to change a partition other than partition 0, choose option (1) and you will be prompted for changes to other partitions, as shown below.

```
Current Fixed Disk Logical Drive dsk/0

 Partition Status  Type Start  End Cyls Sectors
>        0      N UNIX     90  304  215   14609 (7.4 MB)
         1      N    0      0    0    0       0
         2      N    0      0    0    0       0
         3      A  DOS      0   89   90    6120 (3.1 MB)

Total Disk                 0  304  305   20729 (10.0 MB)

If this is not acceptable, type the partition number [0-3] of
   the partition you wish to modify; type "u" if you would
   like to reread the table from the disk; otherwise type "y"
   if okay:
```

The "u" option means "undo"; use it to start over if you make a mess. It re-reads the table from the disk.

Option (3) cycles through each UNIX partition specified and writes to disk only when you have approved everything.

(4) Disk Bad Block Handling

Option (4) allows you to do all the following:

— examine the current bad block list;

— perform sparing of bad blocks;

— change the percentage of the root file system to be used for bad blocks, if the default (1%) is not sufficient. This option is available as part of Standard System Initialization only. (To change the size of the swap area once Opus5 is up and running, use **/etc/swap**(1M).)

Opconfig gives you a choice of each of these as options.

Sparing is done automatically during the standard system initialization procedure for partition-based logical drives.

— the total number of spare blocks (default 1% of the total number of blocks in the drive);

— the file system type (default 3);

— the root file system size in blocks;

— the drive and section number of the device where the root file system resides (`Rootdev`, default drive 0, section 0);

— the drive and section number of the pipe device (`Pipedev`, default drive 0, section 0). Must be the same as `Rootdev`.

— the drive and section number of the dump device (`Dumpdev`, default drive 0, section 0). Must be the same as `Rootdev`.

— the drive and section number of the swap device (`Swapdev`, default drive 0, section 0).

— the block number of the first block in the swap area (`Swplo`);

— the number of blocks in the swap area (`Nswap`, default 4000).

Using Opconfig's Script Facility

To create a script for customer use, first prepare a system with your standard hardware configuration, just as you expect a customer to start with. Boot DOS, and type:

```
C>opmon opsash; :opconfig -O :script
```

Opconfig will start. Answer all questions appropriately. Your answers will be stored in the DOS file *script*. Then, running **opconfig** with the "-I" option using the file *script* should perform identically to the session that created *script*.

To verify that the script is correct, type the following:

```
C>opmon opsash; :opconfig -I :script -P
```

(5) UNIX File System Layout

Option (5) allows you to examine and alter the Opus5 file system layout, including the size and boundaries of your Opus5 file system(s).

By default, in the standard system initialization, one file system is created in the Opus5 disk partition. This is sufficient for most installations. Each disk device (`dsk/0`, `dsk/1`, etc.) is made up of 8 "sections", **/dev/dsk/0s0** through **/dev/dsk/0s7**, **/dev/dsk/1s0** through **/dev/dsk/1s7**, and so on. Additional file systems can be created on default section boundaries, though extra file systems are not usually required. Section 7 is reserved for use by the system.

You can use *opconfig* to examine the current section boundaries, and modify the section boundaries to suit your requirements.

(6) Opus5 UNIX System Parameters

Option (6) allows you to examine and modify a number of kernel system parameters. Each set of parameters is associated with a logical disk drive.

WARNING: Modifying some of these parameters can be extremely dangerous. If you make a mistake, you can do irreversible damage to your file systems. Please be sure you know what you are doing.

The following parameters can be modified:

— the system nodename for UUCP;

— the number of system buffers;

— the name of the bootstrap program (default **:/opus/opsash**);

— the Opus5 boot name (default **/unix**);

— the number of blocks in the current disk drive (`Blocks this drive`);

— the block number of the first spare block (`First spare block`);

This will run through the program and show you the answers that have been stored for each question. At each point before continuing, it waits for you to respond with a carriage return. If you find an incorrect response, run **opconfig** with "-O" again.

Finally, to re-create the end-user environment, type:

```
C>opmon opsash; :opconfig -I :script -D
```

NAME

opdos - interface to DOS

SYNOPSIS

opdos [**-s**] [**-k**] [**-d**] command [options]

DESCRIPTION

Opdos provides an interface to DOS. It allows you to perform operations on DOS files from within the Opus5 operating system. Its main use is within shell procedures; the file **/opus/bin/dos** is a good example.

With the *-s* parameter, *opdos* performs its operations silently. It returns exit codes to the calling process (usually the shell) but does not print error messages.

The *-k* parameter performs a **sync**(1) call to ensure the integrity of the Opus5 file system before executing the *opdos* command.

The *-d* parameter causes buffered output on **/dev/console** to be completely printed before executing the *opdos* command.

Commands and options are the following:

asciiread *filename*

Reads a DOS file deleting newline (**\n**) characters and converting ASCII return (**\r**) to newline (**\n**) for compatibility with Opus5. Characters are read until a **^Z** (0x1a) is encountered or there are no more characters in the file. *Filename* is the name of a DOS file. See *ducp*(1*).

asciiwrite *filename*

Writes a file to DOS in 1024-byte chunks, and converts ASCII newline (**\n**) to **\r\n** for compatibility with DOS. *Filename* can be any file. See *udcp*(1*).

cd *directory name*

Changes the current DOS directory to *directory name.*

chdrv [*drive*]

Changes the current default disk drive for DOS to *drive* where *drive* is 0 for A:, 1 for B:, etc. If *drive* is not specified, it defaults to 0 (A:).

chmod *filename attribute*

Changes the access permissions of a DOS file named *filename*. *Attribute* is one of the following:

00 Readable/Writable (default)
01 Read-only
02 Hidden File
04 System File
08 Volume Label
10 Subdirectory
20 Archive Bit

See DOS documentation for further information.

cmdline

Fetch the DOS command line that was used to invoke the currently running **opmon**.

command

Read a command from DOS for execution under Opus5. See Chapter 5.

del *filename*

Deletes the DOS file *filename*.

diskfree [*drive*]

Reports disk usage on DOS drive *drive*. Default is current drive. *Drive* is specified as follows:

0 current default drive
1 drive A
2 drive B
3 drive C
4 drive D

: etc.
.

Diskfree returns four fields:

number of sectors per cluster
number of clusters currently unused
total number of clusters on the disk
number of bytes per sector

For example, the display

8 9 248 512

means there are 8 sectors per cluster, 9 clusters currently unused, 248 total clusters, and 512 bytes per sector.

You can use these numbers to compute total bytes on the disk, total bytes free, etc.

diskname <drive> [DOS_filename]

"**diskname** <drive>" — without the *DOS_filename* parameter — returns the current DOS file name associated with Opus5 disk drive number <drive>. If *DOS_filename* is included, then the current DOS file name for <drive> is changed to the *DOS_filename* specified.

diskparm <drive>

Returns the Opus5 drive number (0-15 decimal) associated with the <drive> specified. The drive number is the first one or two digits in the filename for the drive; e.g., in **/dev/dsk/0s4**, the drive number is 0.

exit [*exit code*]
exit! [*exit code*]

Exits Opus5 and returns to DOS. From DOS, the command *errorlevel* can be used to read the *exit code*. If no *exit code* is specified, it defaults to 0. **Exit** works only from the console; **exit!** from anywhere.

filelen *filename*
Returns the length of a DOS file *filename* in bytes on standard output.

getmod *filename*
Reports on standard output the access permissions of the DOS file *filename*, according to the *attribute* specifications listed under **chmod** above. These attributes can be OR'd.

isdir *DOS name*
Writes 1 on standard output if *DOS name* is the name of a DOS directory; writes 0 if not.

mkdir *dirname*
Creates a DOS directory named *dirname*.

pause [*exit code*]
pause! [—]
Causes **opmon** to spawn a DOS task. The exit code specifies the exact operation for **opmon** to perform:

0 run DOS shell (default **command.com**)
1 run **opx.bat** with DOS shell
2 run contents of **oparg** with DOS shell
3 run contents of **oparg** directly

Pause works only from the console; **pause!** from anywhere. If no *exit code* is specified, it defaults to 0.

read *filename*
Like **asciiread** without any character conversion. Always reads the whole DOS file. See *ducp*(1*).

rename *oldname newname*
Changes the name of a DOS file from *oldname* to *newname*.

rmdir *dirname*
Removes the DOS directory named *dirname*.

wildcard *string*

Does a wildcard expansion of *string* according to DOS wildcard expansion rules. *String* must be quoted to protect from the UNIX shell. If *string* is a DOS pathname, pathname separators can be either slashes or backslashes. By default **opdos** generates expansions in lower-case. If *string* contains at least one upper-case character and no lower-case characters, **opdos** generates upper-case expansions.

write *filename*

Like **asciiwrite** without conversion. See *udcp*(1*).

DIAGNOSTICS

Diagnostic messages are printed unless the *-s* option is used. With *-s*, only the exit code is returned. Messages and exit codes are the following:

Code	Message	Explanation
1	`no arguments`	*Opdos* called with no arguments.
3	`missing argument`	Option requires at least one argument.
3	`illegal command`	*Opdos* cannot parse your input.
99	`dos: [cmd] failed, error 1`	Invalid function #
98	`dos: [cmd] failed, error 3`	File not found
97	`dos: [cmd] failed, error 3`	Path not found
96	`dos: [cmd] failed, error 4`	Too many open files (no handles left)
95	`dos: [cmd] failed, error 5`	Access denied
94	`dos: [cmd] failed, error 6`	Invalid handle
93	`dos: [cmd] failed, error 7`	Memory control blocks destroyed
93	`dos: [cmd] failed, error 8`	Insufficient memory
91	`dos: [cmd] failed, error 9`	Invalid memory block
90	`dos: [cmd] failed, error 10`	Invalid environment
89	`dos: [cmd] failed, error 11`	Invalid format

88	`dos: [cmd] failed, error 13`	Invalid access code
87	`dos: [cmd] failed, error 13`	Invalid data
85	`dos: [cmd] failed, error 15`	Invalid drive specified
84	`dos: [cmd] failed, error 16`	Attempted to remove the current directory
83	`dos: [cmd] failed, error 17`	Not same device
83	`dos: [cmd] failed, error 18`	No more files

SEE ALSO

dos(1*), ducp(1*), udcp(1*), opmon(1*).

NAME

opload - program to load Opus5 files

SYNOPSIS

/opus/bin/opload

DESCRIPTION

Opload is an Opus-supplied, menu-driven program for loading Opus5 files from diskettes onto your system. It is meant to be used during the initial installation of Opus5. It should also be used for installing any Opus update releases and, in case of a catastrophic disk crash, to restore Opus5 from scratch. Once you have installed files with *opload*, Opus5 is on your fixed disk and needs only to be booted for day-to-day use.

Opus5 files are arranged in groups on the Opus release diskettes. Each group is listed in the *opload* main menu. Some groups are required; these should be installed first. Most groups are optional. Choose the groups you think you will require for your applications. For each group you choose, *opload* describes the group and tells you why you might or might not want to install it. Appendix D contains a description of each group and its size in the current release.

Opload can also delete groups of files, add single files or sets of files, and load Opus update diskettes. A "set" of files is a functional group such as the UUCP subsystem. The "examine" option allows you to determine which group a single file belongs to; you can also **grep**(1) the directory **/opus/contents.<rev level>** for the same information. "<rev level>" means the revision level of the Opus5 software; for example, the directory **/opus/contents.C0** contains the contents of the C0 release of Opus5.

DIAGNOSTICS

Invalid response; please try again. Type <cr> to continue:

You typed a response not understood by *opload*. Review the options available to you at this point in the

program, press the <return> key and try again.

FILES

/opus/contents.<rev level>/*

NAME

opmon - Opus monitor program

SYNOPSIS

C>**\opus\opmon** [*config_file*/**c**]
[*arg_file*/**a**] [*n*/*t*] [*bootfile*]
[; arg list]

DESCRIPTION

Opmon is the monitor software supporting the Opus532 coprocessor system. It runs in the PC under DOS revision 2.0 or greater.

Opmon performs three main functions:

- Opus32 initialization
- Opus32 testing
- Runtime device management

The following options may be given:

config_file
The system configuration file. Defaults to **\opus\opus.cfg**.

arg_file The pathname of the **oparg** file, used to execute UNIX commands from DOS. The default path is **\opus\oparg**.

n The level of memory testing you wish to be performed at bring-up time. Possible values are 1 (medium-level testing, the default), 2 (heavy testing), and 0 (light testing). Medium testing does all memory testing except the "add shifting 1's and 0's". Heavy testing does the "add shifting 1's and 0's"; this can take substantially longer than medium testing. Light testing tests only the first 64 Kbytes of memory.

bootfile A DOS-based file in Opus5 *a.out* format, which is loaded into the Opus16 and started automatically. If no file is specified, *opmon* displays a menu of possible choices (see

below). The boot file must be the last argument in the *opmon* command line before the semi-colon (preceding the *arg list*), or the end of the line.

; arg list

Arguments to the loaded program. The argument list can specify either Opus5 or DOS files. DOS files in the argument list should start with a drive designator and a colon (e.g., c:), to distinguish them from Opus5 files, or a colon to specify the default DOS drive. See examples below.

Examples of *opmon* commands follow.

The command

```
C>opmon opsash; :fsck /dev/dsk/0s0; /unix
```

tells *opmon* to load the *a.out*-format file *opsash* (Opus standalone shell) and start it automatically. It specifies no pause file or configuration file, so defaults are used. When *opsash* is loaded, it automatically does a file system consistency check (loading *fsck* from the DOS default drive) and loads *unix*.

The command

```
C>opmon c:\opus\nucfg/c opsash;/unix
```

tells *opmon* to use the DOS file c:\opus\nucfg for the configuration file, start *opsash*, and load *unix* from the Opus5 file system.

If no options are specified, *opmon* displays "Opus Systems", does a PC Monitor test, and then displays a question mark. Typing carriage return causes the following menu to be displayed:

```
b = DOS debug breakpoint
e = turn error reporting on/off
n = send NMI interrupt
p = pause to DOS
```

```
q = quit to DOS
s = display status byte
t = turn tracing on/off
v = display version
1 = level 1 test

Select one; <space> to do nothing:
```

Some of the options provide a direct hardware interface; these options are meant to be used solely for debugging purposes by experts. Only options "e" and "v" might be of interest to ordinary users.

One of *opmon*'s main functions is to control the kernel's access to all peripheral devices. The kernel sees *opmon* as a very intelligent peripheral device controller. *Opmon* handles requests from the kernel for device access.

DIAGNOSTICS

See "System Messages" chapter.

SEE ALSO

opdos(1*), opsash(1*).

NAME

opsash - Opus standalone shell

SYNOPSIS

C>**opsash**

DESCRIPTION

Opsash is a standalone shell used in diagnosing and booting Opus5. This shell can run a limited set of Opus5 commands found in the DOS directory **opus**, or the Opus5 directory /**stand**. It can also load and run the files **opus****unix** (from DOS) and /**unix**. It recognizes as DOS files any filenames that start with a colon or with a letter and a colon (for example, `c:filename`).

The standalone shell prompt is two dollar signs ($$). Within the standalone shell, # OR backspace erases one character; <cntl-X> OR @ deletes the current line.

Opsash performs the Level 2 selftest. See Chapter 7 for a list of the Level 2 test messages.

To return to DOS from *opsash*, type `dos QUIT`.

SEE ALSO

opmon(1*).

NAME

opunix - DOS program to execute UNIX commands

SYNOPSIS

C> **opunix** [**-i**] [**-o**] [**-b**] [**-v**]
[UNIX command] [UNIX command options]

DESCRIPTION

Opunix provides a mechanism to execute UNIX commands when running DOS and UNIX simultaneously. Pathnames for UNIX files are relative to /. *Opunix* requires that **/opus/bin/udiodaemon** be running under UNIX, and that DOS have been entered with a `dos BG` command.

Opunix accepts DOS STDIN as input and DOS STDOUT as output. This allows the use of the DOS redirection facility.

DOS uses carriage return-line feed (**\r\n**) sequences at the end of lines; UNIX uses ASCII newline characters (**\n**). By default, *opunix* preserves compatibility with DOS by inserting carriage returns and line feeds at the end of lines in UNIX files. This feature can be undesirable if you are transferring or manipulating binary data; thus, this feature can be disabled using the following options:

-i Make a binary copy of the input data; that is, disable DOS compatibility for input data.

-o Make a binary copy of the output data; that is, disable DOS compatibility for output data.

-b Disable DOS compatibility for both input and output data.

The **-v** option displays the version number of *opunix* and exits.

EXAMPLES

The command

```
C>opunix cat /etc/rc > rc
```

does an ASCII copy (preserving DOS compatibility) of the UNIX file /**etc/rc** to the DOS file `rc`.

The command

```
C>opunix -b cat /bin/ls > ls
```

does a binary copy of the UNIX file /**bin/ls** to the DOS file `ls`.

The command

```
C>opunix "cat > opus.cfg" < opus.cfg
```

does an ASCII copy of the DOS file `opus.cfg` to the UNIX file **opus.cfg**. The double quotes preserve redirection within the UNIX command.

The command

```
C>opunix -b "cat > opmon" < opmon.exe
```

does a binary copy of the DOS file `opmon.exe` to the UNIX file **opmon**. The double quotes preserve redirection within the UNIX command.

The command

```
C>opmon sh
```

gives you a UNIX shell prompt. Use ctl-Z <CR> to terminate the UNIX session.

The command

```
C>opunix ls /bin
```

lists the UNIX /**bin** directory.

WARNING

The **udio** channel is not a TTY channel, so commands that do "funny" things to your terminal (**vi**(1), **cu**(1), **pg**(1), etc.) won't work as expected. Signals (DEL, QUIT, etc.) do not work from the keyboard. Ctl-D is not interpreted as end-of-file.

Both UNIX and DOS execute simultaneously when you use *opunix*, so I/O is slower.

SEE ALSO

udiodaemon(1*).

NAME

swap - swap administrative interface

SYNOPSIS

/etc/swap -a swapdev swaplow swaplen
/etc/swap -d swapdev swaplow
/etc/swap -l

DESCRIPTION

Swap provides a method of adding, deleting, and monitoring the system swap areas used by the memory manager. The following options are recognized:

-a Add the specified swap area. *Swapdev* is the name of the block special device, e.g., **/dev/dsk/1s0**. *Swaplow* is the offset in 512-byte blocks into the device where the swap area should begin. *Swaplen* is the length of the swap area in 512-byte blocks. This option can only be used by the super-user. Swap areas are normally added by the system start-up routine **/etc/rc** when going into multiuser mode.

-d Delete the specified swap area. *Swapdev* is the name of the block special device, e.g., **/dev/dsk/1s0**. *Swaplow* is the offset in 512-byte blocks into the device where the swap area should begin. Using this option marks the swap area as "being deleted." The system will not allocate any new blocks from the area, and will try to free swap blocks from it. The area will remain in use until all blocks from it are freed. This option can only be used by the super-user.

-l List the status of all the swap areas. The output has four columns:

DEV The *swapdev* special file for the swap area if one can be found in the **/dev/dsk** or **/dev** directories, and its major/minor

device number in decimal.

LOW The *swaplow* value for the area in 512-byte blocks.

LEN The *swaplen* value for the area in 512-byte blocks.

FREE The number of free 512-byte blocks in the area. If the swap area is being deleted, this column will be marked **(indel)**.

OPUS5 USAGE NOTES

This command requires that *raw* disk space be available for the swap area. That is, if **mkfs** has been done over an area, and any part of that area is being used and cannot be destroyed, it is not available for use as an extra swap area. Thus, if you plan to use this command, you should make sure you have some completely unused disk space. You cannot use extra space in an existing Opus5 file system as a swap area.

The easiest way to use this command is to make each swap device a separate logical disk from Opus5's point of view — either a partition, a DOS file, or a DOS file in a RAMDISK. For example, suppose you decide to set up a swap device as **/dev/dsk/2s0**. Suppose this device is a DOS file residing on disk C: and that it's 5000 512-byte blocks in length. You must modify **opus.cfg** using **opconfig**, and add a DSK device as follows:

```
<dsk/2=fil2(5000,,)>
```

This creates a file called **\opus\opfs\opfil2** on the default hard disk, which will be associated with the Opus5 device **/dev/dsk/2s0**. See Section 6.2.3 of this manual for information on this command format and defaults.

Then, after Opus5 has been brought to single user mode, do a **mknod** command for the device, if it is not already listed in **/dev/dsk**. For example, for **/dev/dsk/2s0**, the **mknod** looks like this:

```
mknod /dev/dsk/2s0 b 0 16
```

Then issue the **swap** command as follows:

```
/etc/swap -a /dev/dsk/2s0 0 5000
```

You might want to add this line to **/etc/rc** to create this swap area each time you go to multi-user mode.

Adding a RAMDISK swap area is exactly like adding a DOS file, except you must be sure to specify the filename of the DOS file on the RAMDISK, including the disk specification. For example, if your RAMDISK is disk D: and your file is called **\swap**, you must say:

```
<dsk/2=fil2(5000,d:\swap,)>
```

If you have more than one swap area, the kernel uses each one in succession in a round-robin fashion. You might not want this, especially if you are using a RAMDISK. You need to do two things to ensure that the RAMDISK is used as much as possible:

1. Make sure the RAMDISK swap area is listed first when you do a **swap -l** command.
2. Call **unix** with the "-a" option in **unix.bat**.

To make sure the RAMDISK is listed first in the swap area table, first verify that it is not listed first by issuing a "swap -l" command. If the RAMDISK is listed first, go on to modify **unix.bat**. If the RAMDISK is not listed first, note the size and starting address of the swap area that is listed first. The strategy is to delete this area and re-create it, so remember these numbers.

Then issue the appropriate "swap -a" command to add the RAMDISK, delete the first swap area using "swap -d", and re-create the former first swap area using the information you remembered.

The "-a" option to the **unix** command in the DOS file **unix.bat** allows you to turn off this alternation of the use of swap areas. Change the line in **unix.bat** that calls **unix** from

```
opmon opsash; :fsck /dev/dsk/0s0; /unix
```

to

```
opmon opsash; :fsck /dev/dsk/0s0; /unix -a
```

WARNINGS

No check is done to see if a swap area being added overlaps with an existing swap area or file system.

SEE ALSO

unix(1*).

NAME

tape - Opus tape handler program

SYNOPSIS

/opus/bin/tape [**-s**] [**-f** <tape path>]
[-bsize] command

DESCRIPTION

Tape is the Opus tape handler program. Its options are as follows:

-s Suppress all messages. The exit code is still sent to indicate whether the function was completed successfully. Messages are output by default.

-f Use *tape path* as the pathname of the tape device. The default *tape path* is **/dev/rmt/0mn** for all commands except *bufin* and *bufout*, in which case the default is **/dev/rmt/0m**.

-b*size* Use *size* as the buffer size for the *bufin* and *bufout* commands. Size is specified in bytes. The default *size* is 256K bytes.

Command can be one of the following:

rew Rewind the tape drive.

fsf Search forward until an end of file mark is passed.

bsf Search backwards until an end of file mark is passed.

fsr <arg>
Skip *arg* records forward. If *arg* is missing, then the default is one record.

bsr <arg>
Skip *arg* records in reverse. If *arg* is missing, then the default is one record.

wfm Write an end of file mark on the tape.

retension
Retension the tape (only usable on cartridge tape drives).

erase Erase the whole tape (only usable on cartridge tape drives).

status Return status. See DIAGNOSTICS below.

bufout Read standard input into a large buffer and then rapidly write it to the tape. Used on cartridge tapes so they will operate in "streaming" mode. For example, to create a **cpio**(1)-format tape using **find**(1) and **tape bufout**, use

```
find . -print | cpio -ocvB | tape bufout
```

bufin Rapidly read the tape into a large buffer and then write the buffer to standard output. Used on cartridge tapes so they will operate in "streaming" mode. For example, to read in a **cpio**-format tape using **bufin**, use

```
tape bufin | cpio -icvdumB
```

DIAGNOSTICS

All commands except *bufin* and *bufout* print out a hex number if they complete. This number is the status of the tape drive at the conclusion of the requested operation. Bits in the status word are as follows:

bit 0 Tape write-protected.

bit 1 Tape drive offline.

bit 2 Beginning of tape.

bit 3 End of tape.

bit 4 End of file.

bit 5 Error.

bit 6 Command not implemented.

bit 7 Command timed out.

NAME

udcp - Opus5 to DOS copy

SYNOPSIS

udcp [-a] Opus5_file DOS_file

DESCRIPTION

Udcp copies an Opus5 file to a DOS file. With the *-a* option, it performs an *opdos asciiwrite* operation on the Opus5 file, so that ASCII newline characters are converted to **\r\n** sequences for compatibility with DOS. With no option, it performs an *opdos write*, so no character conversion is done.

Udcp requires that both filenames be specified explicitly. To specify DOS pathnames from within Opus5, use two backslashes wherever you would ordinarily use one within DOS; this is necessary because the backslash \ is a special character for the shell. For example,

INCORRECT: `udcp /usr/file \opus\file`
CORRECT: `udcp /usr/file \\opus\\file`

In Opus5 releases C3 and later, *udcp* allows DOS-style wildcard substitutions. Note that these differ from UNIX-style wildcard conventions, in some important ways, notably that "*" can be used only at the *end* of a pattern.

DIAGNOSTICS

Same as *opdos*(1*).

SEE ALSO

ducp(1*), opdos(1*).

NAME

udiodaemon - provides a listener process under UNIX for *opunix* commands

SYNOPSIS

/opus/bin/udiodaemon

DESCRIPTION

Udiodaemon is typically invoked by **/etc/rc**. It forks a shell for every request that comes from *opunix*. The shell is executed to be owned by root, with a home directory of /.

SEE ALSO

opunix(1*).

NAME

unix - Opus5 kernel

SYNOPSIS

C>**opmon opsash; /unix**

DESCRIPTION

Unix is the Opus5 kernel, which is booted by **opmon** in the DOS batch file **\opus\unix.bat**. Ordinarily, you would run **opconfig** to set up the system configuration, and boot the system using the standard **unix.bat** file. In the standard **unix.bat**, **/unix** is called with no arguments.

But the **/unix** command in **unix.bat** can also take a number of options that overwrite the parameters set by **opconfig**. This means you can temporarily set or change system parameters without running **opconfig**.

Using the options described here is seldom desirable or necessary. Note that these options can also be EXTREMELY DANGEROUS. You can do IRREVERSIBLE DAMAGE to your file systems if you make a mistake. Please use extreme caution.

The options are as follows:

-a Swap allocation. If you have more than one swap area, the kernel by default uses each one in succession in a round-robin fashion. You might not want this if one of your swap areas is on a fast disk or a RAMDISK. To inhibit this, make sure your fast disk is the first one listed in the swap area table, and call **/unix** with this option. See **swap**(1M) in this Appendix.

-b *num* The number of system buffers.

-i *dix* The device index number of the device containing the root file system.

-k *devnum* The number of the dump device *dumpdev*.

-l *block #* The address of the first block in the swap area, *swaplo*.

-m *pages* The maximum number of 1024-byte pages in physical memory.

-n *nodename*
The system nodename for UUCP.

-p *devnum* The number of the pipe device *pipedev*. Must be the same as the **-r** option (*rootdev* device).

-r *devnum* The number of the root device *rootdev*.

-S *#blocks* Sets the number of blocks in *swaplim*. The largest process running is sent SIGTERM if the free swap space is less than *swaplim*. Default is off.

-s *devnum* The number of the swap device *swapdev*.

-u *#blocks* The maximum file size *ulimit*.

-w *#blocks* The number of 512-byte blocks in the swap area (*nswap*).

-z *drive section start len*
Modified information about a section of a logical disk.

Numerical parameters can be input in decimal, hex (preceded by 0x), or octal (preceded by 0).

The device number *devnum* is computed by multiplying the major device number by 256 and adding the minor device number. Since the major device number in these cases is usually 0 (the block **dk** device), *devnum* is usually just the minor device number.

EXAMPLES

To make the nodename "opus", change the line in the DOS file **unix.bat** from

```
opmon opsash; :fsck /dev/dsk/0s0; /unix
```

to

```
opmon opsash; :fsck /dev/dsk/0s0; /unix -n
opus
```

To also increase the maximum file size, do the following:

```
opmon opsash; :fsck /dev/dsk/0s0; /unix -n
opus -u 10000000
```

FILES

/unix

SEE ALSO

opmon(1*)

NAME

dial - establish an outgoing terminal line connection

SYNOPSIS

```
#include <dial.h>

int dial (call)
CALL call;

void undial (fd)
int fd;
```

DESCRIPTION

Dial(3C*) is the Opus-enhanced version of the standard *dial*(3C) function, which it supersedes. The function returns a file-descriptor for a terminal line open for reading and writing. Read the manual page for **dial**(3C) in conjunction with this one.

The Opus function differs from the standard *dial* function in that it allows you to supply the name of your own customized dialing program in the "call devices" field of **/usr/lib/uucp/L-devices**. This works as follows. First, an application program that dials via a modem must fill in the CALL structure from **dial.h**. Then it must call *dial*. These steps are shown below:

```
/* myprog to perform dialing */

#include <dial.h>

        char devused [64];
        CALL mycall;
        int dial();
        ...

main()
{
        int fd;
        mycall.attr = NULL;
```

```
        mycall.speed = mycall.baud = 1200;
        mycall.line = "tty128";
        mycall.device = devused;
        mycall.dev_len = 64;
        mycall.telno = "4155551212";
        ...

        fd = dial(mycall);
        ...
}
```

At this point, the *dial* procedure automatically goes to the file **/usr/lib/uucp/L-devices** and looks for a line in that file that starts with "ACU" and matches both the `line` and `baud` elements. For example, if you use the Opus-supplied program **/opus/bin/hayes** as the dialing program, a line for dial-outs in **/usr/lib/uucp/L-devices** might look like the following:

```
ACU tty128 /opus/bin/hayes 1200
```

The **dial** procedure checks to see if a program name is in the third field, and if one exists and is executable, it opens the dial-out line, sets its attributes, and **exec**'s the program with the following arguments:

arg[0] the name by which the dialer program was invoked (in this example, "`/opus/bin/hayes`")

arg[1] value of `mycall.device`. The "`call.device`" field exists in **dial** for the special case where **dial** is called with a NULL `call.line` parameter, which means the user doesn't care which line is used. In this case, **dial** uses the first line that matches the specified baud rate, and returns the name of the line in the `device` field. So the field is simply a pointer to a character array that will contain the name of the device actually used.

arg[2] `atoi(mycall.baud)`. **Dial** interprets a "-1" in this field as an indication that the user doesn't care about the baud rate, and in this case, returns in `call.baud` the baud rate actually used.

arg[3] `mycall.telno`

arg[4] `atoi(mycall.speed)`

arg[5] `atoi(mycall.modem)`

arg[6] `mycall.line`

where "`mycall.`" refers to an element of the CALL structure passed in to **dial**. See **dial**(3C) for more information on the CALL structure.

An application program need not use all the parameters passed to it by *dial*; for example, **/opus/src/hayes.c** uses only five of them.

When **dial** invokes your dialing program, it returns either the file descriptor of the file it opened, or the exit code with which your program terminates. Legal exit codes and their meanings can be found in **/usr/include/dial.h**.

Opus provides source and object for a sample customized dialing program, which works for Hayes and Hayes-compatible modems. The source is **/opus/src/hayes.c**; the object is **/opus/bin/hayes**.

FILES

/usr/lib/uucp/L-devices
/usr/spool/uucp/LCK...*tty-device*

SEE ALSO

dial(3C)

NAME

doschdir - change the current DOS directory

SYNOPSIS

doschdir(fname)
char *fname;

DESCRIPTION

Doschdir allows access from Opus5 to DOS function hex 3B. It changes the current DOS directory to *fname*.

RETURN VALUE

If successful, returns 0.

DIAGNOSTICS

Returns -1 if an error occurs, at which point **doserrno** can be examined. If an Opus5 error occurs in trying to use **/dev/dos**, *doserrno* will contain a value greater than 0, which will be a regular **UNIX** error number (see Introduction to Section 2 of the *UNIX System V UNIX System V Programmer Reference Manual*). If a DOS error occurs, *doserrno* will contain, as a negative number, the regular DOS error number. For example, the DOS error number 6 (bad handle) would be in *doserrno* as -6.

NAME

doschdrv - change the default DOS drive

SYNOPSIS

doschdrv(drive)
int drive;

DESCRIPTION

Doschdrv allows access from Opus5 to DOS function hex E. It makes *drive* (0=A, 1=B, etc.) the new default DOS drive.

RETURN VALUE

If successful, returns the total number of drives (diskette plus fixed). DOS always assumes there are 2 diskette drives (A: and B:). Thus, if the system has only one physical diskette drive, it is nevertheless counted as two.

DIAGNOSTICS

Returns -1 if an error occurs, at which point **doserrno** can be examined. If an Opus5 error occurs in trying to use **/dev/dos**, *doserrno* will contain a value greater than 0, which will be a regular **UNIX** error number (see Introduction to Section 2 of the *UNIX System V Programmer Reference Manual*). If a DOS error occurs, *doserrno* will contain, as a negative number, the regular DOS error number. For example, the DOS error number 6 (bad handle) would be in *doserrno* as -6.

NAME

doschmod - change the attributes of a DOS file

SYNOPSIS

doschmod(fname, func, attributes)
char *fname;
int func;
int attributes;

DESCRIPTION

Doschmod allows access from Opus5 to DOS function hex 43. It can read or change the following attributes of a DOS file *fname*: read-only, hidden, system, and archive.

Func determines if the current access permissions are to be changed or simply reported. If *func* is 1, the access permissions will be changed to the value in *attributes*; if *func* is 0, then the file's current *attributes* are returned.

Attributes are specified according to current DOS conventions; see your DOS documentation.

RETURN VALUE

If successful, and *func* is 0, returns the current *attributes*.

DIAGNOSTICS

Returns -1 if an error occurs, at which point **doserrno** can be examined. If an Opus5 error occurs in trying to use **/dev/dos**, *doserrno* will contain a value greater than 0, which will be a regular **UNIX** error number (see Introduction to Section 2 of the *UNIX System V Programmer Reference Manual*). If a DOS error occurs, *doserrno* will contain, as a negative number, the regular DOS error number. For example, the DOS error number 6 (bad handle) would be in *doserrno* as -6.

NAME

dosclose - close a DOS file handle

SYNOPSIS

dosclose(handle)
int handle;

DESCRIPTION

Dosclose allows access from Opus5 to DOS function hex 3E. It closes the file designated by *handle* and flushes all internal buffers in DOS.

Handle is the file handle returned by *dosopen*.

DIAGNOSTICS

Returns -1 if an error occurs, at which point **doserrno** can be examined. If an Opus5 error occurs in trying to use **/dev/dos**, *doserrno* will contain a value greater than 0, which will be a regular **UNIX** error number (see Introduction to Section 2 of the *UNIX System V Programmer Reference Manual*). If a DOS error occurs, *doserrno* will contain, as a negative number, the regular DOS error number. For example, the DOS error number 6 (bad handle) would be in *doserrno* as -6.

NAME

doscreat - create a DOS file

SYNOPSIS

doscreat(fname, attributes)
char *fname;
int attributes;

DESCRIPTION

Doscreat allows access from Opus5 to DOS function hex 3C. It creates a DOS file of name *fname* with the attributes specified by *attributes*. If a file named *fname* already exists, it is truncated to zero length in preparation for writing the new file.

Attributes are specified according to current DOS conventions; see your DOS documentation.

RETURN VALUE

If successful, returns the new file handle.

DIAGNOSTICS

Returns -1 if an error occurs, at which point **doserrno** can be examined. If an Opus5 error occurs in trying to use **/dev/dos**, *doserrno* will contain a value greater than 0, which will be a regular **UNIX** error number (see Introduction to Section 2 of the *UNIX System V Programmer Reference Manual*). If a DOS error occurs, *doserrno* will contain, as a negative number, the regular DOS error number. For example, the DOS error number 6 (bad handle) would be in *doserrno* as -6.

NAME

dosdiskfree - report DOS disk free space

SYNOPSIS

```
dosdiskfree(drive, sectperclust, numclust,
      totclust, bytespersect)
int drive;
int *sectperclust, *numclust, *totclust,
      *bytespersect;
```

DESCRIPTION

Dosdiskfree allows access from Opus5 to DOS function hex 36. It reports disk usage on DOS drive *drive*. Default is the current drive. *Drive* is specified as follows:

0	current default drive
1	drive A
2	drive B
3	drive C
4	drive D
	etc.

RETURN VALUE

Dosdiskfree returns four values:

`*sectperclust`
: number of sectors per cluster

`*numclust`
: number of clusters currently unused

`*totclust`
: total number of clusters on the disk

`*bytespersect`
: number of bytes per sector

You can use these numbers to compute total bytes on the disk, total bytes free, etc.

DIAGNOSTICS

Returns -1 if an error occurs, at which point **doserrno** can be examined. If an Opus5 error occurs in trying to use **/dev/dos**, *doserrno* will contain a value greater than 0, which will be a regular **UNIX** error number (see

Introduction to Section 2 of the *UNIX System V Programmer Reference Manual*). If a DOS error occurs, *doserrno* will contain, as a negative number, the regular DOS error number. For example, the DOS error number 6 (bad handle) would be in *doserrno* as -6.

NAME

dosioctl - I/O control for DOS files

SYNOPSIS

dosioctl(handle, func, buf, len)
int handle;
int func;
char *buf;
int len;

DESCRIPTION

Dosioctl allows access from Opus5 to DOS function hex 44. The parameters mean the following:

handle — For *func* values 0, 1, 2, 3, 6, 7, and 10, *handle* is a DOS file handle. For other values of *func*, *handle* is a device number (0 = default, 1 = A, 2 = B, ...).

func — The function to be performed.

buf — For *func* values 2, 3, 4, and 5, this is a pointer to a buffer of data which is *len* long. For *func* values 0, 1, 9, 10, and 11, it is a pointer to a value of type short.

len — For *func* values, 2, 3, 4, and 5, this is the length of *buf* in bytes. For *func* value 11, it is the number of times to execute the delay loop while retrying a function.

DIAGNOSTICS

Returns -1 if an error occurs, at which point **doserrno** can be examined. If an Opus5 error occurs in trying to use **/dev/dos**, *doserrno* will contain a value greater than 0, which will be a regular **UNIX** error number (see Introduction to Section 2 of the *UNIX System V Programmer Reference Manual*). If a DOS error occurs, *doserrno* will contain, as a negative number, the regular DOS error number. For example, the DOS error number 6 (bad handle) would be in *doserrno* as -6.

NAME

doslseek - move the file read/write pointer

SYNOPSIS

doslseek(handle, offset, method)
int handle;
int offset;
int method;

DESCRIPTION

Doslseek allows access from Opus5 to DOS function hex 42. *Handle* is the file handle returned by *dosopen*. Offset is the offset in bytes, and *method* is one of the following:

0 The pointer is moved to *offset* bytes from the beginning of the file.

1 The pointer is moved to the current location plus offset.

2 The pointer is moved to the end-of-file plus offset. This method can be used to determine the file's size.

RETURN VALUE

If successful, *doslseek* returns the new location of the pointer.

DIAGNOSTICS

Returns -1 if an error occurs, at which point **doserrno** can be examined. If an Opus5 error occurs in trying to use /**dev/dos**, *doserrno* will contain a value greater than 0, which will be a regular **UNIX** error number (see Introduction to Section 2 of the *UNIX System V Programmer Reference Manual*). If a DOS error occurs, *doserrno* will contain, as a negative number, the regular DOS error number. For example, the DOS error number 6 (bad handle) would be in *doserrno* as -6.

NAME

dosmkdir - create a DOS directory

SYNOPSIS

dosmkdir(dirname)
char *dirname;

DESCRIPTION

Dosmkdir allows access from Opus5 to DOS function hex 39. If *dirname* is not a fully qualified pathname, *dosmkdir* creates a DOS directory of name *dirname* at the end of the current path.

RETURN VALUE

If successful, returns 0.

DIAGNOSTICS

Returns -1 if an error occurs, at which point **doserrno** can be examined. If an Opus5 error occurs in trying to use **/dev/dos**, *doserrno* will contain a value greater than 0, which will be a regular **UNIX** error number (see Introduction to Section 2 of the *UNIX System V Programmer Reference Manual*). If a DOS error occurs, *doserrno* will contain, as a negative number, the regular DOS error number. For example, the DOS error number 6 (bad handle) would be in *doserrno* as -6.

NAME

dosopen - open a DOS file and return the DOS handle

SYNOPSIS

dosopen(fname, mode)
char *fname;
int mode;

DESCRIPTION

Dosopen allows access from Opus5 to DOS function hex 3D. *Fname* is the name of the DOS file. *Mode* is input according to current DOS conventions; see your *DOS Technical Reference Manual* for details.

RETURN VALUE

Dosopen returns the file handle, a binary value that you must use in order to refer to the file once it has been opened (like a file descriptor in **UNIX**).

DIAGNOSTICS

Returns -1 if an error occurs, at which point **doserrno** can be examined. If an Opus5 error occurs in trying to use **/dev/dos**, *doserrno* will contain a value greater than 0, which will be a regular **UNIX** error number (see Introduction to Section 2 of the *UNIX System V Programmer Reference Manual*). If a DOS error occurs, *doserrno* will contain, as a negative number, the regular DOS error number. For example, the DOS error number 6 (bad handle) would be in *doserrno* as -6.

NAME

dosread - read from a DOS file or device

SYNOPSIS

dosread(handle, buf, len)
int handle;
char *buf;
int len;

DESCRIPTION

Dosread allows access from Opus5 to DOS function hex 3F. It reads *len* bytes from the file designated by *handle*, using buffer address *buf*. *Handle* is the file handle returned by *dosopen*.

RETURN VALUE

If successful, returns the number of bytes read.

DIAGNOSTICS

Returns -1 if an error occurs, at which point **doserrno** can be examined. If an Opus5 error occurs in trying to use **/dev/dos**, *doserrno* will contain a value greater than 0, which will be a regular **UNIX** error number (see Introduction to Section 2 of the *UNIX System V Programmer Reference Manual*). If a DOS error occurs, *doserrno* will contain, as a negative number, the regular DOS error number. For example, the DOS error number 6 (bad handle) would be in *doserrno* as -6.

NAME

dosrename - rename a DOS file

SYNOPSIS

dosrename(oldname, newname)
char *oldname;
char *newname;

DESCRIPTION

Dosrename allows access from Opus5 to DOS function hex 56. It renames a DOS file from *oldname* to *newname*.

If a drive name is used in the specification of *newname*, it must be the same as the drive specified or implied in *oldname*. The directory paths need not be the same, allowing a file to be moved to another directory and renamed in the process.

RETURN VALUE

If successful, returns 0.

DIAGNOSTICS

Returns -1 if an error occurs, at which point **doserrno** can be examined. If an Opus5 error occurs in trying to use **/dev/dos**, *doserrno* will contain a value greater than 0, which will be a regular **UNIX** error number (see Introduction to Section 2 of the *UNIX System V Programmer Reference Manual*). If a DOS error occurs, *doserrno* will contain, as a negative number, the regular DOS error number. For example, the DOS error number 6 (bad handle) would be in *doserrno* as -6.

NAME

dosrmdir - remove a DOS directory

SYNOPSIS

dosrmdir(fname)
char *fname;

DESCRIPTION

Dosrmdir allows access from Opus5 to DOS function hex 3A. It removes a DOS directory of name *fname*. The current DOS directory cannot be removed. An error is returned if the specified directory is not empty.

RETURN VALUE

If successful, returns 0.

DIAGNOSTICS

Returns -1 if an error occurs, at which point **doserrno** can be examined. If an Opus5 error occurs in trying to use **/dev/dos**, *doserrno* will contain a value greater than 0, which will be a regular **UNIX** error number (see Introduction to Section 2 of the *UNIX System V Programmer Reference Manual*). If a DOS error occurs, *doserrno* will contain, as a negative number, the regular DOS error number. For example, the DOS error number 6 (bad handle) would be in *doserrno* as -6.

NAME

dosunlink - delete a file from a specified directory

SYNOPSIS

dosunlink(fname)
char *fname;

DESCRIPTION

Dosunlink allows access from Opus5 to DOS function hex 41. It removes the directory entry associated with *fname*.

Read-only files cannot be deleted by this call. To delete a read-only file, first call *doschmod* to change the file's attribute to 0, and then delete the file with *dosunlink*.

RETURN VALUE

If successful, returns 0.

DIAGNOSTICS

Returns -1 if an error occurs, at which point **doserrno** can be examined. If an Opus5 error occurs in trying to use **/dev/dos**, *doserrno* will contain a value greater than 0, which will be a regular **UNIX** error number (see Introduction to Section 2 of the *UNIX System V Programmer Reference Manual*). If a DOS error occurs, *doserrno* will contain, as a negative number, the regular DOS error number. For example, the DOS error number 6 (bad handle) would be in *doserrno* as -6.

NAME

doswrite - read from a DOS file or device

SYNOPSIS

doswrite(handle, buf, len)
int handle;
char *buf;
int len;

DESCRIPTION

Doswrite allows access from Opus5 to DOS function hex 40. It writes *len* bytes from the file designated by *handle*, starting at buffer address *buf*. *Handle* is the file handle returned by *dosopen*.

RETURN VALUE

If successful, returns the number of bytes written.

DIAGNOSTICS

Returns -1 if an error occurs, at which point **doserrno** can be examined. If an Opus5 error occurs in trying to use **/dev/dos**, *doserrno* will contain a value greater than 0, which will be a regular **UNIX** error number (see Introduction to Section 2 of the *UNIX System V Programmer Reference Manual*). If a DOS error occurs, *doserrno* will contain, as a negative number, the regular DOS error number. For example, the DOS error number 6 (bad handle) would be in *doserrno* as -6.

NAME

getdiskname - get name of specified disk

SYNOPSIS

```
getdiskname(drive, name)
int drive;
char *name;
```

DESCRIPTION

Getdiskname gets the name used by *opmon*(1*) for disk drive *drive* (e.g., "c:10", "fil0"). This name is the one in the file **opus.cfg**(4*).

RETURN VALUE

If successful, returns the name in *name*.

DIAGNOSTICS

Returns 0 if successful or a standard **UNIX** error number (see Introduction to Section 2 of the *UNIX System V Programmer Reference Manual*).

NAME

getdiskparm - get full 16 bits of drive number

SYNOPSIS

getdiskparm(drive, parm)
int drive;
int *parm;

DESCRIPTION

The "drive_number" parameter to DK device specifications in **opus.cfg** is a 16-bit value. The low-order 4 bits are used for the drive number (0-15), but the other bits can also be turned on by the user. *Getdiskparm* gets the full 16 bits of the "drive_number" field for the drive specified by *drive* (0-15).

RETURN VALUE

If successful, returns the 16-bit drive number parameter in *parm*.

DIAGNOSTICS

Returns 0 if successful or a standard **UNIX** error number (see Introduction to Section 2 of the *UNIX System V Programmer Reference Manual*).

NAME

setdiskname - get name of specified disk

SYNOPSIS

setdiskname(drive, name)
int drive;
char *name;

DESCRIPTION

Setdiskname sets the name to be used by *opmon*(1*) for disk drive *drive* (e.g., "c:10", "fil0"). The new name is specified in *name*. This name follows the format specifications for the file **opus.cfg**(4*). The new name will be used the next time *opmon* opens the disk drive.

DIAGNOSTICS

Returns 0 if successful or a standard **UNIX** error number (see Introduction to Section 2 of the *UNIX System V Programmer Reference Manual*).

NAME

syscmdline - fetch the DOS command line

SYNOPSIS

syscmdline(buf, len)
char *buf;
int len;

DESCRIPTION

Syscmdline fetches the DOS command line that was used to invoke the currently running **opmon**.

RETURN VALUE

If successful, returns the command line in *buf*, and the count in bytes of the number of characters in the command line as the return value.

DIAGNOSTICS

Returns -1 if an error occurs, at which point **doserrno** can be examined. If an Opus5 error occurs in trying to use **/dev/dos**, *doserrno* will contain a value greater than 0, which will be a regular **UNIX** error number (see Introduction to Section 2 of the *UNIX System V Programmer Reference Manual*). If a DOS error occurs, *doserrno* will contain, as a negative number, the regular DOS error number. For example, the DOS error number 6 (bad handle) would be in *doserrno* as -6.

NAME

sysexit - exit to DOS

SYNOPSIS

sysexit(exitcode)
int exitcode;

DESCRIPTION

Sysexit exits Opus5, terminates **opmon**, and goes into DOS.

From DOS, the command **errorlevel** can be used to read the *exitcode*.

RETURN VALUE

If successful, does not return.

DIAGNOSTICS

Returns -1 if an error occurs, at which point **doserrno** can be examined. If an Opus5 error occurs in trying to use **/dev/dos**, *doserrno* will contain a value greater than 0, which will be a regular **UNIX** error number (see Introduction to Section 2 of the *UNIX System V Programmer Reference Manual*). If a DOS error occurs, *doserrno* will contain, as a negative number, the regular DOS error number. For example, the DOS error number 6 (bad handle) would be in *doserrno* as -6.

NAME

syspause - pause to DOS

SYNOPSIS

syspause(exitcode)
int exitcode;

DESCRIPTION

Syspause suspends Opus5 and goes into DOS.

The exit code is passed to **opmon**; see **opdos**(1*).

RETURN VALUE

If successful, returns 0.

DIAGNOSTICS

Returns -1 if an error occurs, at which point **doserrno** can be examined. If an Opus5 error occurs in trying to use **/dev/dos**, *doserrno* will contain a value greater than 0, which will be a regular **UNIX** error number (see Introduction to Section 2 of the *UNIX System V Programmer Reference Manual*). If a DOS error occurs, *doserrno* will contain, as a negative number, the regular DOS error number. For example, the DOS error number 6 (bad handle) would be in *doserrno* as -6.

NAME

sysreadio - read contents of the PC bus

SYNOPSIS

sysreadio (adx)
unsigned int adx;

DESCRIPTION

Sysreadio reads the contents of the PC bus I/O port at *adx* and returns this byte value (0-255).

DIAGNOSTICS

Returns -1 if an error occurs, at which point **doserrno** can be examined. If an Opus5 error occurs in trying to use **/dev/dos**, *doserrno* will contain a value greater than 0, which will be a regular **UNIX** error number (see Introduction to Section 2 of the *UNIX System V Programmer Reference Manual*). If a DOS error occurs, *doserrno* will contain, as a negative number, the regular DOS error number. For example, the DOS error number 6 (bad handle) would be in *doserrno* as -6.

NAME

sysreadmem - read contents of the PC bus memory

SYNOPSIS

```
sysreadmem(segment, offset, buf, len)
unsigned short segment;
unsigned short offset;
unsigned char *buf;
unsigned int len;
```

DESCRIPTION

Sysreadmem reads the contents of PC bus memory starting at *segment:offset* into the buffer *buf* for *len* bytes.

DIAGNOSTICS

Returns 0 if successful or -1 if the call fails, at which point **doserrno** can be examined. If an Opus5 error occurs in trying to use **/dev/dos**, *doserrno* will contain a value greater than 0, which will be a regular **UNIX** error number (see Introduction to Section 2 of the *UNIX System V Programmer Reference Manual*). If a DOS error occurs, *doserrno* will contain, as a negative number, the regular DOS error number. For example, the DOS error number 6 (bad handle) would be in *doserrno* as -6.

NAME

syswriteio - write a value to the PC bus

SYNOPSIS

syswriteio(adx, val)
unsigned int adx;
unsigned char val;

DESCRIPTION

Syswriteio writes *val* into the PC bus I/O port at *adx*.

DIAGNOSTICS

Returns 0 if successful or -1 if the call fails, at which point **doserrno** can be examined. If an Opus5 error occurs in trying to use **/dev/dos**, *doserrno* will contain a value greater than 0, which will be a regular **UNIX** error number (see Introduction to Section 2 of the *UNIX System V Programmer Reference Manual*). If a DOS error occurs, *doserrno* will contain, as a negative number, the regular DOS error number. For example, the DOS error number 6 (bad handle) would be in *doserrno* as -6.

NAME

syswritemem - write to the PC bus memory

SYNOPSIS

syswritemem(segment, offset, buf, len)
unsigned short segment;
unsigned short offset;
unsigned char *buf;
unsigned int len;

DESCRIPTION

Syswritemem writes the contents of *buf* for *len* bytes into the PC bus memory starting at *segment:offset*.

DIAGNOSTICS

Returns 0 if successful or -1 if the call fails, at which point **doserrno** can be examined. If an Opus5 error occurs in trying to use **/dev/dos**, *doserrno* will contain a value greater than 0, which will be a regular **UNIX** error number (see Introduction to Section 2 of the *UNIX System V Programmer Reference Manual*). If a DOS error occurs, *doserrno* will contain, as a negative number, the regular DOS error number. For example, the DOS error number 6 (bad handle) would be in *doserrno* as -6.

NAME

opus.cfg - system configuration file

DESCRIPTION

Opus.cfg is a system configuration file in the DOS directory \opus that is normally created and maintained by **opconfig**(1*). **Opmon**(1*) reads this file at initial startup. The file defines the environment in which Opus5 is to operate, including the following:

- segment number: memory address of Opus16 coprocessor
- interrupt level of Opus16 coprocessor
- local timezone
- attention key
- available I/O devices
- clock rate

In **opus.cfg**, only text inside of [square] or <angle> brackets is significant; anything else is ignored. Square brackets indicate system parameters; angle brackets indicate devices.

Segment Number — The address segment number set in the hardware installation procedure must be specified. The standard configuration file contains the following line:

```
[s=?]
```

This means that the Opus software searches segments 8, 9, A, D, and E. If it can't find the Opus coprocessor board using one of these segments, it issues an error message. On the PC AT, segment C replaces segment E.

On the TI Professional Computer, the Opus software searches segments 8, 9, A, and B.

For standard systems, this setting need not be changed.

Substitute a hex number for "?" if:

- you have reason to believe the board will not be found using the search method outlined above; or
- for security reasons, you do not want the Opus software to check other segments; or
- you want to place the Opus segment at a non-standard address.

Example: `[s=9]`

For further details of switch settings, see Section 8.2.3 (Segment Number).

Interrupt Level — The interrupt level set in the hardware installation procedure must be specified. The standard configuration file contains the following line:

```
[i=x]
```

This means operation without interrupts.

If you would like operation with interrupts, change `[i=x]` to correspond to the switch setting on the Opus16 coprocessor board. For details of switch settings, see Section 8.2.2 (Interrupt Request Level).

In general, it is never necessary to use any interrupt level other than level 7.

TI Professional Computers users should see Appendix A for special instructions.

Time Zone — Change the time zone according to the directions in Section 4.3.

Attention Key — The attention key sequence puts you into **opmon** any time you hit it from within any Opus5 program. The default setting is ALT-128, or

```
[a=128]
```

If you use ALT-128 for some other purpose, you should change the attention key setting.

The numbers 1, 2 and 8 must be entered from the numeric keypad. On the TI Professional Computer, the character code is sent when you've typed three characters preceded by ALT. Thus, on the TI PC, you can type ALT-1, ALT-2, and ALT-8 separately; that is, you don't have to hold down the ALT key. A consequence of this for TI PC users is that you must type three characters, even if you don't need three; for example, you must type ALT-002 to get ALT-2. On the IBM and other PCs, the character code is sent when you let up on the ALT key. Thus you can type ALT-2, let up on ALT, and "002" will be sent.

Default input is decimal; hex input can be preceded by "0x". Here are some guidelines for changing the attention key sequence.

1. The sequence ALT-<base 10 number less than 256 when all digits are entered at the numeric keypad> produces the same binary sequence as <hex of the base 10 number>. For example, typing ALT-65 gives the same result as typing capital A (whose hex code is 41 = 65 decimal); it produces a capital A. Since 7-bit ASCII uses all 128 possible characters, you probably want to avoid attention sequences lower than 128 decimal; i.e., do not use ALT-0 through ALT-127, since these have other meanings. You can, however, use any code that uses the eighth bit, e.g., codes 128 through 255. For example,

   `[a=255]` or `[a=0xFF]` = ALT-255

2. If you need to use all possible characters in 8-bit ASCII (codes ALT-0 through ALT-255), you can still use ALT-<function key code>,

because the function codes are not part of 8-bit ASCII. For example,

```
[a=0x7100]
```

makes the attention key sequence ALT-F10. See PC documentation for the function key codes.

3. If you need to recognize every possible key code (8-bit ASCII plus all combinations of function key codes), then you should turn off the attention capability. Do this with

```
[a=-1]
```

This probably won't inconvenience you very much; you can use the DOS interface from the shell to reach into **opmon**.

Devices — This set of entries specifies the peripheral devices available on the system. If you are using devices not listed in the standard **opus.cfg**, you must add these devices.

See Chapter 6 for information on devices.

Opus.cfg must specify the PC real-time clock rate if it is different from the defaults (40 Hz for TI PCs, 18.2 Hz for all others). **Opconfig**(1*) handles this automatically for most kinds of PCs during standard system initialization.

SEE ALSO

opmon(1*).