OVERVIEW OF THE XENIX* 286 OPERATING SYSTEM

*XENIX is a trademark of Microsoft Corporation.

Intel Corporation, 3065 Bowers Avenue, Santa Clara, California 95051

Order Number: 174385-001

OVERVIEW OF THE XENIX* 286 OPERATING SYSTEM

Order Number: 174385-001

*XENIX is a trademark of Microsoft Corporation.

Intel Corporation, 3065 Bowers Avenue, Santa Clara, California 95051

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

Intel Corporation makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Intel Corporation assumes no responsibility for any errors that may appear in this document. Intel Corporation makes no commitment to update or to keep current the information contained in this document.

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

The following are trademarks of Intel Corporation and its affiliates and may be used only to identify Intel products:

BITBUS	i_m	iRMX	Plug-A-Bubble
COMMputer	iMDDX	iSBC	PROMPT
CREDIT	iMMX	iSBX	Promware
Data Pipeline	Insite	iSDM	QUEST
Genius	int_el	iSXM	QueX
i	int_elBOS	Library Manager	Ripplemode
i	Intelevision	MCS	RMX/80
I^2ICE	$int_eligent$ Identifier	Megachassis	RUPI
ICE	$int_eligent$ Programming	MICROMAINFRAME	Seamless
iCS	Intellec	MULTIBUS	SLD
iDBP	Intellink	MULTICHANNEL	SYSTEM 2000
iDIS	iOSP	MULTIMODULE	UPI
iLBX	iPDS	OpenNET	

Microsoft, MS-DOS, Multiplan, and XENIX are trademarks of Microsoft Corporation. UNIX is a trademark of Bell Laboratories. CP/M is a trademark of Digital Research Incorporated. VAX is a trademark of Digital Equipment Corporation.

REV.	REVISION HISTORY	DATE
-001	Original issue	11/84

CONTENTS

CONTENTS **PAGE**

CONTENTS **PAGE**

FIGURES

intel®

CHAPTER 1
INTRODUCTION TO XENIX

Audience

This overview is intended for new XENIX users who want a basic knowledge of XENIX and for experienced users who want a list of commands and programming tools. This overview is the XENIX manual you should read first. It introduces you to the XENIX operating system and to the full set of XENIX manuals. After you have read this manual, you should understand what an operating system is, be familiar with basic XENIX concepts and terminology, have an overall view of what is included in the system, and understand what information is presented in each manual in the set of XENIX manuals.

Chapters

This overview has these chapters and appendixes:

1. **Introduction to XENIX** -- an introduction to the *Overview of the XENIX 286 Operating System*. It describes the book's intended audience and chapters, presents operating system concepts, and briefly describes the history and features of the XENIX operating system.

2. **Files and File Systems** -- description of ordinary files, directories for organizing files, special files (devices), file protections, tools for working with files, and the new file system.

3. **Running Programs** -- introduction to programs and processes. This chapter explains what happens when you run programs, from system startup to login to executing commands.

4. **Text Processing** -- brief description of tools for people who prepare documents for printing or typesetting.

5. **Programming** -- introduction to XENIX programming concepts, including the C programming language, standard function libraries, system calls, supporting tools, shell programming, and customizing XENIX.

A. **Basic System Commands** -- brief definitions of the commands in the Basic System.

B. **Text Formatting Commands** -- brief definitions of the text formatting commands in the Extended System.

C. **Programming Tools** -- brief definitions of the commands, libraries, and system calls in the Extended System.

D. **Related Publications** -- a list of related Intel publications and suggested readings.

The Basic System and the Extended System

Intel has divided the XENIX operating system into two different products to satisfy different user requirements. These products are the Basic System and the Extended System.

The Basic System

The Basic System is intended for all users. It has all of the things needed to run application software and to administer the system. It also has general-purpose tools like the **ed** and **vi** text editors, electronic communications, and many commands. These manuals accompany the Basic System:

- *Overview of the XENIX 286 Operating System.* This manual is intended for all XENIX users. It describes operating systems in general and XENIX in particular, covering important concepts such as files and file systems, the shell, and commands. Programming tools are introduced for programmers, and text processing features are described for writers and text processors.

- *XENIX 286 Installation and Configuration Guide.* This manual is for the system administrator. It gives complete instructions for installing XENIX software from $5\frac{1}{4}$-inch and 8-inch flexible disks. The section on configuration explains how to add devices to the system and remove devices from it.

- *XENIX 286 User's Guide.* The user's guide is intended for all users of the system. It has brief tutorials that introduce basic concepts and common commands, and it has full chapters on the **ed** text editor, **vi** text editor, electronic mail, Bourne shell (**sh**), and **bc** calculator.

- *XENIX 286 Visual Shell User's Guide.* This guide explains how to use the visual shell, which is a user interface based on menus. The menus list common functions and application software programs that the system administrator has added.

- *XENIX 286 System Administrator's Guide.* This manual is for the system administrator. It describes the procedures that the system administrator performs on a regular basis, such as administering users, making back-up copies of files, and monitoring system use.

- *XENIX 286 Communications Guide.* This manual is for the system administrator. It explains how to set up and administer a Micnet or **uucp** communications network.

- *XENIX 286 Reference Manual.* This manual is intended for anyone who wants technical information or a detailed list of options for different commands. It summarizes the syntax and options of each command in the Basic System. It also has reference information about files and devices.

The Extended System

The Extended System is made up of software development tools and text formatting tools. The software development tools include utility programs, standard C libraries, system calls, a C compiler, an assembler, a linker, a loader, a debugger, a lexical analyzer, and a compiler-compiler (a program that generates a compiler). The text formatting tools include commands for improving writing, **mm** (memorandum) macros, and standard **nroff** and **troff** programs. The **mm** macros are codes that you use to prepare memos, letters, reports, and other documents. The **nroff** program formats documents for a printer, and the **troff** program formats documents and prints them on a phototypesetter.

These manuals are part of the Extended System:

- *XENIX 286 Programmer's Guide.* This manual is intended for applications programmers. It describes these important programming tools: **cc** (C compiler), **lint** (C program checker), **make** (a program maintainer), SCCS (a source code control system), **adb** (a program debugger), **as** (an assembler), **lex** (a lexical analyzer generator), **yacc** (yet another compiler-compiler), and **m4** (a macro processor). Appendixes discuss C language portability and give reference pages for programming commands.

- *XENIX 286 C Library Guide.* This manual is intended for programmers. It describes system calls and standard libraries of C subroutines. It covers standard I/O functions, screen processing, character and string processing, process control, pipes, signals, system resources, and error processing. Appendixes give reference information such as the assembly language interface, programming differences in this release, and reference pages for individual subroutines, system calls, and file formats.

- *XENIX 286 Device Driver Guide.* This manual is intended for a programmer who writes device drivers. Chapters cover the kernel, simple character device drivers, terminal device drivers, block device drivers, instructions for adding drivers to the configuration, designing and debugging hints, and drivers supplied with XENIX. Appendixes give related reference information.

- *XENIX 286 Text Formatting Guide.* This manual is intended for writers who want to prepare manuscripts for printing or phototypesetting. It gives an overview of text processing, describes writing and editing tools, explains how to use macros, **nroff**, and **troff**, and shows how to format tables and mathematics.

What Is an Operating System?

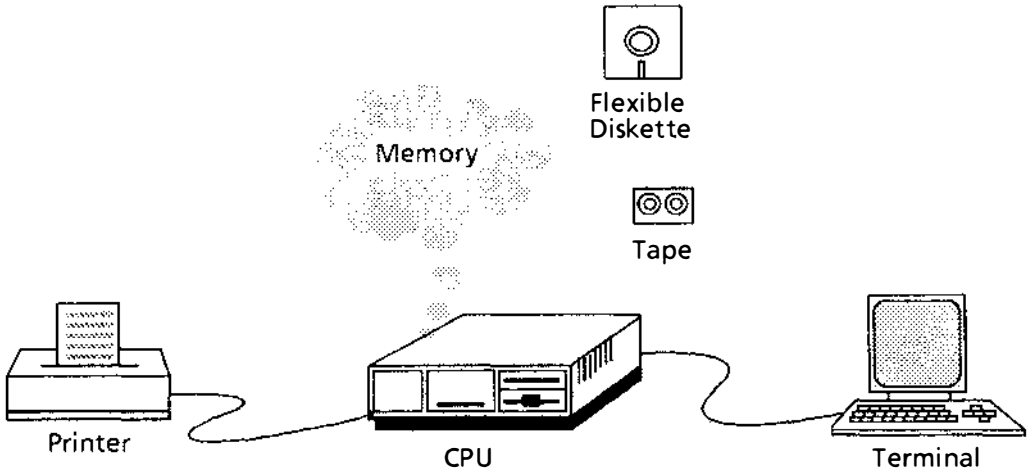
An operating system is a set of programs that manage the hardware resources of a computer and provide useful services. It has three basic components: the kernel, a set of utility programs, and a command interpreter.

When you want to work on a computer, you need to send data from one device to another. For example, if you are writing a letter at your terminal, you need to store it on a disk. Later, you may want to print it on a printer. To complicate matters, someone else may want to use the printer at the same time. Clearly, the resources of the computer system have to be shared. These are some of the reasons that the computer has an operating system.

Hardware Devices

Since the operating system coordinates the activities of the hardware, it is useful to identify the functions that different pieces of hardware perform.

A typical computer system has a CPU (central processing unit) plus several hardware devices, such as terminals, disks, memory, printers, and tape drives (see Figure 1-1).



F-0321

Figure 1-1. Hardware Devices

The hardware devices serve these functions:

- CPU. The CPU does all of the processing. It reads instructions one by one and executes them, performs necessary logic operations, and makes mathematical calculations.

- Terminal. The terminal is the device you use to communicate with the computer. It has a keyboard so that you can enter information and a screen that displays what you type.

- Disks. Disks store programs and data for fast and easy retrieval.

- Memory. Memory is an area where data is stored while it is processed.

- Printers. Printers produce a copy of data on paper.

- Tape drives. Tape drives store copies of programs and data on tape.

The Kernel

The kernel is a software program that interacts directly with computer hardware. When the system administrator starts the computer, the kernel is loaded into memory from disk storage. It remains there as long as the computer is running and oversees all of the activities of the computer system. When you give commands or run application software, these programs may use system calls to ask for services from the kernel. For example, each time a program wants to read a file, it sends a **read** system call to the kernel.

In a multiuser computer system, several people share memory, printers, and other computer resources. It is the kernel that gives each person exclusive use of a resource for a period of time. The kernel's function is to do this so efficiently that users are unaware that the resources are being shared.

A kernel typically performs these functions:

- Mass storage management. A computer system stores a large amount of data on disks. The kernel maintains some form of file system on disks to keep track of all this data so that it can be located and used.

- Process management. In a computer system, many processes may be running at the same time. The kernel gives each process a share of processing time and keeps track of each process.

- Memory management. Any computer system has a certain amount of memory and that memory often has to be shared by several processes. The kernel gives each process an area in memory and keeps one process from interfering with another. If a process needs more memory than is available, the kernel temporarily moves the process out of memory and onto disk until it is time to bring it back into memory. This technique is called swapping.

- Device management. Each device in a computer system has special characteristics that the kernel has to understand to send information to and from devices. In the kernel, software programs called device drivers communicate with devices. When a program uses a system call, the kernel selects the appropriate device driver.

- Error checking. The kernel constantly checks the operation of the system and displays error messages when problems occur.

- Accounting. A multiuser operating system normally keeps some kind of records of how resources have been used. These records may be the basis for billing for computer time or for evaluating computer use.

Utility Programs

Some people speak of the kernel as the operating system, but the kernel is usually accompanied by a set of utility programs that you can run to create files, copy files, and perform other useful functions. Throughout this manual, these programs are considered part of the operating system.

One type of program that comes with an operating system is a text editor, which is a tool that you use to type programs, reports, and other text. A text editor has commands for adding, changing, and deleting lines of text.

Other programs that are usually available are programming tools, such as standard software libraries, compilers, linkers, loaders, and assemblers. Tools for checking and debugging code are sometimes included.

A Command Interpreter

You request services from the operating system by giving commands. Every operating system has at least one command interpreter that takes your commands so programs can be executed.

What Is the XENIX 286 Operating System?

XENIX 286 is Intel's value-added version of the XENIX operating system released by Microsoft Corporation. Microsoft's XENIX, in turn, is a value-added version of the System III UNIX operating system developed by Bell Laboratories at AT&T. XENIX 286 also includes features developed by the University of California at Berkeley. XENIX 286 supports multiple users and multiple tasks. It has all of the components of standard operating systems--a kernel, utility programs, and a command interpreter--with features that many others do not have.

The XENIX Kernel

The XENIX kernel performs all of the functions that a typical operating system kernel performs. It manages mass storage, processes, memory, and devices, and it checks for errors during operation. The system administrator can turn process accounting logs on or off and clear them as desired.

The XENIX kernel has these important features:

- Standard system calls. Programmers can use over 60 different system calls to request services from the kernel. These system calls include all those provided by UNIX System III.

- Speed. The speed of the kernel is driven by the speed of the processor, and the XENIX 286 system runs on Intel's iAPX 286 microprocessor, which represents the leading edge of microprocessor technology.

- Small size. The XENIX kernel has only 10,000 or so lines of code.

- High-level language. Most of the kernel is written in the C programming language rather than in assembly language, so the operating system can run on many different computers.

- Hierarchical file system. XENIX has a hierarchical file system so you can organize your files of information. Intel has redesigned the file system to increase processing speed. Chapter 2 describes the new file system.

- Devices treated like files. You can send data to devices and take data from them just as if they were ordinary files. This is called device independence.

- Separate code and data. Program code and data are kept in separate areas of memory, which is efficient since several users may share one copy of the code.

- Buffer management. Buffers are areas where data is stored when it is brought from a disk. With Microsoft's exported buffer management technique, the kernel has access to buffers that are outside its primary data segment. With Intel's enhancements, programs are loaded directly into memory, and buffer contents are left undisturbed. Programs are thus loaded faster and information in the buffers can continue to be used.

- Device drivers. Intel releases XENIX 286 with device drivers for terminals, Winchester disks, flexible disks, and tapes, plus a guide to writing device drivers so you can add appropriate hardware devices to your system.

Utility Programs

Together, the Basic System and the Extended System offer over 200 utility programs. You can create additional functions without writing C programs if you take advantage of tools called pipes (they connect the output of one program with the input of another) or write shell programs using a shell command interpreter.

The utility programs are all stored on a disk, and you run them by giving commands to a command interpreter called a shell. In many discussions of XENIX, the terms "command", "program", and "utility" are used interchangeably.

The utility programs for the Basic System are listed in Appendix A, and the utility programs for the Extended System are listed in Appendix B and Appendix C.

Command Interpreters

A XENIX command interpreter is called a shell. You communicate with the operating system by typing commands that the shell interprets. For example, if you want the operating system to print a calendar, you give the **cal** command and the shell responds to it.

In some operating systems, the command interpreter is part of the kernel and cannot be changed easily, but in XENIX it is a separate C program that can be modified or replaced by another C program. The Basic System has three different shells: the Bourne shell, restricted shell, and visual shell. The Extended System adds the C shell. These shells are discussed in Chapter 3.

Who Uses the XENIX Operating System?

Everyone on the computer uses the operating system, but people work with it in different ways. Users typically fall into one of these categories: users who run application software, the system administrator, application programmers, systems programmers, and writers and text processors. XENIX 286 gives each of these types of users tools to make their work easier and to improve their final products.

- Users who run application software. Many users run application software, such as word processing or a spreadsheet program. These users may be aware of the operating system only when logging on the computer (logging on is typing your name and giving your password). This is especially true if you use a visual shell that lists applications and functions on a menu. Users who run application software often use XENIX's office tools, such as electronic mail, personal calendars, and a desktop calculator. These tools are in the Basic System.

- System administrators. The system administrator is the person responsible for maintaining the computer and its software. The system administrator needs to understand system operations very well and to know how to install XENIX, add devices, add users, monitor system use, make duplicate copies of data, tailor the environment, solve system problems, and set up communications networks. Administering the system has been simplified by new commands for adding users, removing users, making system backups, and other common tasks. Procedures for the system administrator are outlined in the *XENIX 286 Installation and Configuration Guide*, the *XENIX 286 System Administrator's Guide*, and the *XENIX 286 Communications Guide*. All of these manuals are in the Basic System.

- Application programmers. Application programmers write software such as general ledgers and spreadsheets. Application programmers normally use the operating system's text editors and commands for working with files. To do programming other than shell programming they need the Extended System, which has commands for developing software, libraries of standard functions, system calls, and programming tools. Programmers who develop software on one XENIX system can usually put it on several machines with only minor changes.

- Systems programmers. Systems programmers change the operating system to meet the requirements of a particular product. They add device drivers and add or change utility programs. The *XENIX 286 Device Driver Guide* in the Extended System has instructions for writing device drivers plus examples of different drivers.

- Writers and text processors. Writers and text processors produce documents such as programs, memos, letters, and books. These users can create documents with the XENIX text editors. They need the Extended System to print or typeset documents with standard features such as centering and bolding.

The History of XENIX

XENIX has evolved over more than a decade and has been used successfully in many different environments.

UNIX

The history of XENIX begins with the development of UNIX at AT&T's Bell Laboratories.

In the late 1960s, Ken Thompson and others at Bell Laboratories were participating in a project that involved a large, sophisticated, multiuser operating system called Multics on a large mainframe computer from General Electric Corporation. When Bell Laboratories left the project, Thompson wanted to move a particular program called Space Travel from the mainframe to a dedicated PDP-7 computer, so he created a new operating system for it. Since the new operating system was for single users, it was named UNIX as a play on the name Multics.

The first version of UNIX was written in assembly language. It was a personal effort by a programmer who wanted a system that made it easy to write, test, and run programs. He also favored elegance of design, and the limited size of his development computer encouraged economy and elegance. By 1971, the new operating system was being used within Bell Laboratories on Digital Equipment Corporation's PDP-7 and PDP-9 computers.

The second version of UNIX included software written in a programming language called B. This language was used when UNIX was moved to Digital Equipment Corporation's PDP-11/20 family of minicomputers in 1971. The PDP-11/20 was purchased to support the development of a text formatting package.

The third version of UNIX came in 1973. It was a complete rewriting of the operating system in C, which was a revision of B. C was a good choice for an operating system because it was a high-level, structured language and yet it was able to manipulate small units of data efficiently. This version incorporated multiprogramming, a technique that keeps several programs in memory at once so that the central processing unit is used to advantage. The system ran on several computers in the PDP-11 family.

Since C was a high-level language, UNIX could run on more than one computer. The fourth version of UNIX eliminated all code that was specific to the PDP-11 family of computers. This new version was produced in 1977 and was moved onto the Interdata 8/32, which was quite different from the PDP-11s.

Through the 1970s, UNIX was used mostly within Bell Laboratories, but by 1975 AT&T began to license it, and other research agencies began to work with it. Many colleges have licenses to use UNIX, and many computer scientists have become familiar with it.

Today, several versions of UNIX are in circulation. The first UNIX system to be licensed commercially was Version 7, a multiuser system released in 1978. An update was introduced in 1981 as System III. A subsequent version, System V, was released in 1983. There was no System IV. The name UNIX remains, even though the system now supports multiple users.

UNIX became popular at Bell Laboratories, then gained supporters in research centers and universities, then attracted the attention of software developers and computer manufacturers. It has become popular because of many valuable features, including

- Portability. One feature of UNIX that truly sets it apart from traditional operating systems is portability. Most operating systems have been tied to a specific computer or family of computers because they were written in assembly language that only those computers could use. The UNIX operating system is written almost entirely in C, a high-level language that can run on many different computers. Application software developed on one computer can run on many computers. It is sometimes necessary to make minor changes to the software, but it is not necessary to rewrite much of it.

- Multiuser support. UNIX is a multiuser system, which means that several users can work on a system at one time.

- Multitasking system. UNIX is a multitasking system, which means that several users can run processes simultaneously, and that an individual user can run several processes in the background while working at the terminal. For example, you can edit a program at the terminal while you print a report on the printer.

- UNIX tools. The UNIX philosophy is to provide many small tools that can be changed or combined to perform new functions. You can create new tools without writing C programs by writing shell scripts (files of shell commands, which can include statements from the shell programming language) or by using pipes. Pipes are tools that connect the output of one program with the input of another.

- Office tools. Office tools such as individual calendars, user-to-user communications, and a desktop calculator are all part of the UNIX system.

- Programming tools. UNIX tools have evolved over time in response to specific needs of programmers. As programmers have worked with the system, they have corrected errors, added new features, and created new utility programs.

- Networking. Several UNIX systems can be linked together so that data, including electronic mail, can be sent from one system to another.

- Access to status information. UNIX makes status information readily available so any user can check who is on the system, what processes are running, and what printers are busy.

- Groups. When people work on the same projects, they often need to share files. With UNIX this is encouraged because files can be assigned to groups of users.

- Device independence. Devices, such as printers and terminals, are accessed like files, so you can send data to a device just as you send it to any file. Likewise, you can bring data from a device.

Microsoft's XENIX

Microsoft's XENIX is an enhanced version of UNIX derived from UNIX System III. In earlier releases, valuable features such as interprocess communication with semaphores, performance improvements for microcomputers, and file locking were added. File locking is important because it regulates changes to a file. If one user is changing a part of a file, all other users can be locked out of the file until the change is complete. These enhancements continue to be important to commercial users and to manufacturers who use XENIX on microcomputers.

Intel moved the previous release of XENIX from the 8086 microprocessor to the iAPX 286 microprocessor, and Microsoft used the new product as the basis for its Release 3 of the operating system. Microsoft also added many features that benefit the users who run application software, system administrators, application programmers, systems programmers, and text processors.

New Microsoft Features in the Basic System

The current release of XENIX includes these new features:

- Micnet communications. A new communications package called Micnet has been integrated with a new mailer based upon the Berkeley **mail** program. With Micnet you can send mail between local machines over serial lines, execute remote commands, and transfer files from one machine to another. Although the traditional **uucp** network is still available, this new Micnet network is intended to replace **uucp** for local machine communications.

- System administration commands. Several commands have been added to make system administration easier. For example, a new **sysadmin** command presents options for copying and recovering data on a menu, and a new **acctcom** command prints accounting information.

- Password administration. A new command, **pwadmin**, has been added so the system administrator can force users to change their passwords at regular intervals to reduce the chance that an unauthorized person can discover passwords. The **pwcheck** command has been added so the system administrator can check the **etc/passwd** file.

- Secure startup sequence. The startup sequence prevents a user from going into single-user mode without knowing the system administrator's password.

- Visual shell. A new shell, called a visual shell, has been added for users who find it easier to select menu options than to remember commands. The most common processes are listed on the menu, and the system administrator can add or change menu selections. This user interface is similar to the Multiplan interface.

- Batch execution at a specified time. Earlier releases included the **at** command, which you can use to place commands in a queue and define when they are to be executed. This release adds the **atq** command so you can check the queue and the **atrm** command so you can remove a command from the queue.

- Assignable devices. You can use the **assign** and **deassign** commands to restrict a device for your exclusive use. For example, you may insert a flexible disk in a drive and prevent others from using the drive while it is inserted.

New Text Formatting Features

The Extended System includes these new text formatting features from Microsoft:

- New macros. Macros are codes that you can use to prepare documents for printing or typesetting. The new **mm** memorandum macros are superior to the **ms** macros provided in an earlier release.

- New commands. Writers and text processors can use several new commands to format text, catch formatting errors, and improve the literary quality of documents. They can prepare text with constant width for typesetting with the **cw** and **cwcheck** commands, cut out selected columns of text with **cut**, merge selected columns of text with **paste**, check commands for typesetting mathematical expressions with **eqncheck**, mark differences between text files with **diffmk**, and locate awkward phrases with **diction**.

New Programming Features

The Extended System includes these new programming features from Microsoft:

- Fixed stack analysis utilities. New utility programs analyze C programs to help determine stack size requirements, which is useful for those who develop software for fixed stack machines such as unmapped iAPX 86, iAPX 286, and some M68000 systems.

- MS-DOS file access utilities. Commands that allow MS-DOS files and directories to be read from and written to are available. These commands will be useful to those whose computers can operate both MS-DOS and XENIX. Access to IBM DOS 1.1 and 2.0 format disks is supported.

- Source code control system (SCCS). The new **cdc** command enhances the SCCS system for controlling source code. You use source code control commands to monitor changes to source files. The **cdc** command changes the delta commentary of an SCCS delta.

- System calls. This release of the product has all of the system calls of the previous release (which was based upon UNIX Version 7), all of those in UNIX System III, and several new ones.

  These new system calls let unrelated processes share data: **sdget, sdfree, sdgetv, sdenter, sdleave, sdwaitv**. The **chsize** system call truncates files to a given length, which is important to efficient execution of some FORTRAN write operations. The **nap** system call lets a process sleep for less than one second, which is useful for interactive, screen-oriented software.

- Language tools. This release has a new C compiler that has UNIX System III language extensions and supports large model processes (multiple data segments and text segments). This compiler gives you the option of using the expanded instruction set for the iAPX 286 microprocessor. The release also has **cref** and **xref** commands so you can generate cross-reference listings from C source code.

- Floating-point support. XENIX includes a floating-point emulator and support for the 80287 floating-point hardware.

- Exported buffer management. Buffers are areas in memory that hold data temporarily while the kernel is waiting to use it. Computer systems use buffers to make computer operation more efficient. Data can be kept ready and waiting so that it can be used as soon as the CPU or some device is ready. Microsoft increased the amount of buffer space available by giving the kernel access to buffers in segments outside its own data segment.

Berkeley Enhancements

Researchers at the University of California at Berkeley became involved with UNIX and moved it onto the VAX computer created by Digital Equipment Corporation. As they have worked with the product, they have added features to make UNIX easier to use or to give it additional power. Microsoft's release of the XENIX 286 product has several features developed by the University of California at Berkeley.

These Berkeley commands are included in the Basic System:

- **finger** - find information about users
- **head** - print the first few lines of a file
- **lc** - list directory contents in columns (Berkeley's enhanced directory listing)
- **mail** - send, read, or dispose of mail
- **more** - display information one screen at a time
- **tset** - set terminal modes
- **vi** - invoke the screen-oriented text editor

These Berkeley commands and libraries for programmers are included in the Extended System:

- **csh** - invoke the C shell
- **ctags** - create a tags file for the **ex** and **vi** editors
- **curses** - perform screen and cursor functions
- **dbm** - perform data base functions
- **mkstr** - create an error message file and change C source
- **soelim** - make word processing documents portable to other UNIX-based systems
- **strings** - find the printable strings in a file
- **termcap** - perform device-independent terminal functions
- **xstr** - extract strings from C programs to implement shared strings

Intel's Contribution to XENIX

Intel has entered the XENIX market as a technological leader. Intel invented the microprocessor and offers XENIX on systems with the iAPX 286 microprocessor, one of the fastest microprocessors on the market.

Intel gives its OEMs (original equipment manufacturers) the opportunity to put their XENIX-based products on the best technology at every level of integration--from components to boards to complete systems.

Intel's goal is total performance--performance that is based on the latest technology, the most effective software, and the most successful users:

- Latest technology. Intel's strategy is to combine the latest UNIX-based technology with the latest silicon technology. Its iAPX 286 microprocessor is one of the fastest microprocessors available and memory management has been integrated into the chip.

- Effective software. To improve software efficiency, Intel has created a new file system that reduces the amount of time spent searching for data. It is described in Chapter 2.

  Intel has rewritten the **dump** and **restor** programs so that they do more error checking and give more options. The **restor** command now has an option that verifies that files have been restored successfully. The **dump** command has an option that tells you when the last dumps were made and what the levels were.

- Successful users. Intel recognizes that users need a basic understanding of XENIX plus specific information about functions they perform on their jobs. As a result, the new manual set includes books that give the big picture as well as books that are oriented toward particular users.

  This manual attempts to help you understand the operating system and learn basic concepts and terminology.

  Three books give the system administrator detailed information. The new *XENIX 286 Installation and Configuration Guide* walks the administrator through installing the system and adding and removing devices such as printers, terminals, and disk drives. The new *XENIX 286 System Administrator's Guide* outlines a system administrator's responsibilities and has how-to instructions for overseeing daily operations and solving system problems. The *XENIX 286 Communications Guide* explains how to set up and administer Micnet and **uucp** communications networks.

  The new *XENIX 286 Device Driver Guide* gives systems programmers instructions and examples so that they can create their own device drivers.

  Intel has also simplified the installation process.

intel®

CHAPTER 2
FILES AND FILE SYSTEMS

Ordinary Files

All of the data that you and other users produce is kept in files. Technically, an ordinary XENIX file is just a series of bytes stored on a mass storage device under a specific name. The bytes are regular ASCII text (letters, numbers, and characters such as punctuation marks), or they are binary codes (codes representing information in a form that cannot be displayed directly on a screen).

The Content of an Ordinary File

You create an ordinary file by using a text editor, compiling a program, or running an application program that creates files. It contains only what you put in it. For example, an ordinary file may have a source program, an executable program, a letter, or payroll data. XENIX does not keep record counts or use a special marker to show the end of a file. Figure 2-1 has an example of an ordinary file. Notice that it has nothing but text.

```
MEMO

TO        Team
FROM      Mary
SUBJECT   Revised Schedules

Please give me your revised schedules by Friday.
```

Figure 2-1. Sample Ordinary File

The Structure of an Ordinary File

XENIX does not expect data to be stored in any particular format. It is just text. When you create a file, you may give it a format, then use that format when you write programs. For example, the **/etc/passwd** file has one record for each user. The record has seven fields of information and they are separated by colons. The sample line below illustrates the format of the file. (If you are curious about the meaning of the fields, see Chapter 3.)

```
mary:j9Hz1FzBYSOVw:201:200:M Day,Rm 21,x5006,273-5543:/usr/mary:/bin/sh
```

The XENIX kernel is not aware of this format, but programs that read the **/etc/passwd** file need to understand it.

The Name of an Ordinary File

These are the rules and conventions that apply to file names in XENIX:

- When you want to work with a file, you identify it by name. The kernel keeps track of each file by assigning it a unique number, called an inode number, but it is not necessary for you to use the number.

- The name of a file can have 1 to 14 characters.

- The name can include any keyboard character except a slash (/). However, the recommended procedure is to avoid blanks, invisible characters such as BACKSPACE, and these special characters, which have a special meaning to the command interpreter:

  `-  ?  *  [  ]  "  '  \`

- Both uppercase and lowercase letters can be used, and they are different characters. For example, "Memo.to.Jack" is not the same as "memo.to.jack".

- If a file name begins with a dot, it will not appear on your list of files unless you use a special option of the command that lists files (**ls -a**).

- You may use dots in file names. For example, "memo.to.jack" uses dots. To XENIX, these dots in a name are just characters, but some characters with dots are meaningful. XENIX uses several combinations of a dot and a character at the end of a file name to identify a particular kind of file. These combinations are called suffixes. For example, programmers should use a ".c" suffix for programs they write in the C programming language. These are some of the suffixes that are meaningful to XENIX:

.a	A library archive
.c	A program written in the C programming language
.h	An include file for the C programming language
.l	Input for **lex**
.o	The object code created by a compiler or assembler
.s	A program written in assembly language
.y	Input for **yacc**

- In any directory, a file name must be unique. For example, if you have a directory named "memos", it can have only one file named "memo.to.jack". However, someone else could have a file named "memo.to.jack" in some other directory. Directories will be explained in detail later in this chapter.

The Size of an Ordinary File

When you create a file, you cannot define its maximum size. The file can continue to grow up to a limit of four megabytes as long as the disk has space for more data. The system administrator can increase this maximum size with the **ulimit** command built into the Bourne shell, and anyone can decrease the maximum size with that command.

Directories

As the number of files increases, it becomes important for you to have some way of organizing them so that you can locate them easily. The XENIX solution is to let you organize your own files by creating a hierarchical structure of directories.

A directory is just a list of files and their unique file numbers, which are called inode numbers. The organization of directories is discussed in this section. Inode numbers are explained later in the chapter.

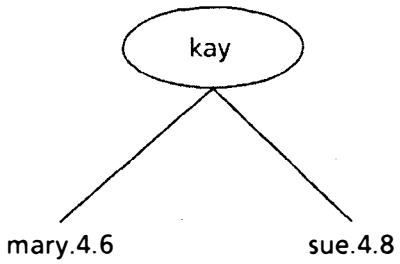
Login Directories

When the system administrator adds you to the system, a login directory is created for you. This is the directory where you will begin each time you work on the computer. It is sometimes called the home directory.

Imagine that the system administrator adds a new user named Kay, gives her the login name "kay", and defines "kay" as her login directory. When she logs in, she is placed in her "kay" directory.

You may place files of information in your login directory, or define subdirectories so that you can organize your files, or both. For example, Kay expects to create few files so she sees no reason to use subdirectories. Since she will have few files, she will keep all of them in her login directory and scan the list when she wants to work with one. Suppose that she creates two memos, "mary.4.6" and "sue.4.8", in her login directory. Figure 2-2 shows what her hierarchy would look like. Figure 2-3 shows what Kay sees when she uses the directory listing command (**ls**) to look at her login directory. The "$" is a standard prompt that means you can give commands.

A hierarchy is often referred to as a tree structure because it looks like an inverted tree with branches.



F-0307

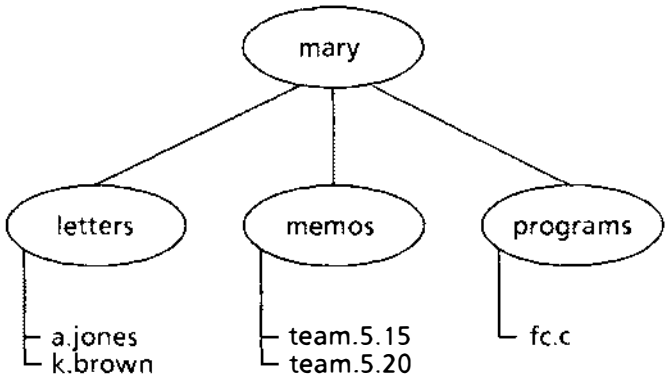
Figure 2-2. Sample Hierarchy of Login Directory without Subdirectories

```
$ ls
mary.4.6
sue.4.8
$
```

Figure 2-3. Sample Contents of Login Directory without Subdirectories

Subdirectories

If you have a large number of files, you may use subdirectories to group related files. A subdirectory is a directory within a directory. For example, imagine that Mary plans to create memos, letters, and programs. She can create three separate directories, "memos", "letters", and "programs" in her login directory, then place files in the appropriate subdirectory. Figure 2-4 shows her hierarchy after she has created two memos, two letters, and a program called "fc.c". Figure 2-5 shows what Mary actually sees when she uses the **ls** (list) command and the **lc -R** command to look at her login directory. The **ls** command shows only the contents of the directory you are in. The **lc -R** command shows the contents of that directory plus the contents of each subdirectory. Notice that the shorthand name (.) is used for the working directory.



F-0308

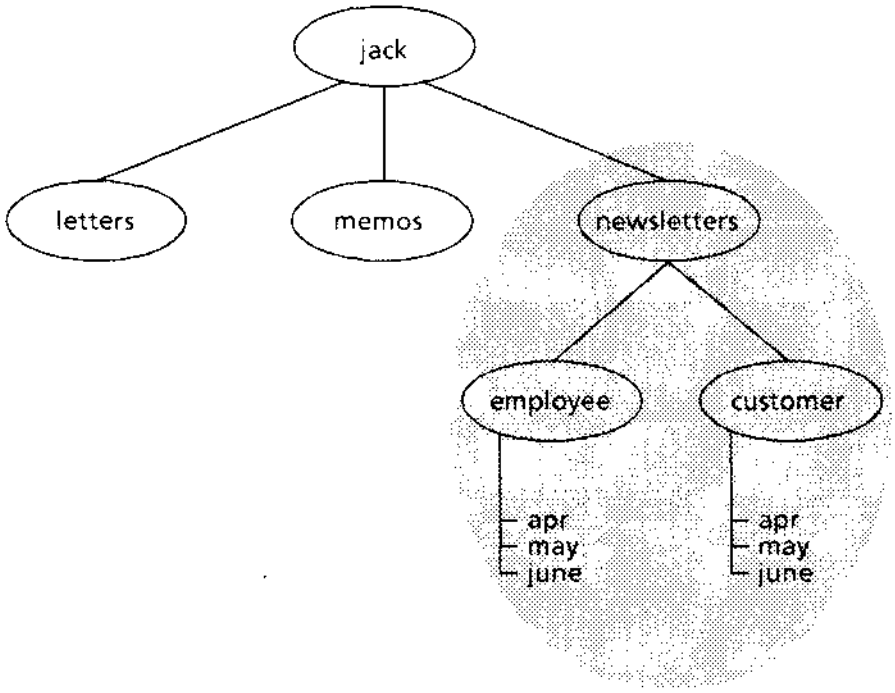
Figure 2-4. Sample Hierarchy for Login Directory with Subdirectories

```
$ ls
letters
memos
programs

$ lc -R
letters                 memos                   programs

./letters:
a.jones                 k.brown

./memos:
team.5.15.              team.5.20

./programs:
fc.c
$
```

Figure 2-5. Sample Directory List

Subtrees

A subtree begins at a directory and includes all of the files and directories under it. For example, imagine that Jack is added to the system and that he creates directories for letters, memos, and newsletters. He wants to keep employee newsletters separate from customer newsletters, so he creates subdirectories for them under his "newsletters" directory. Figure 2-6 shows the "newsletters" subtree of the "jack" hierarchy.



F-0309

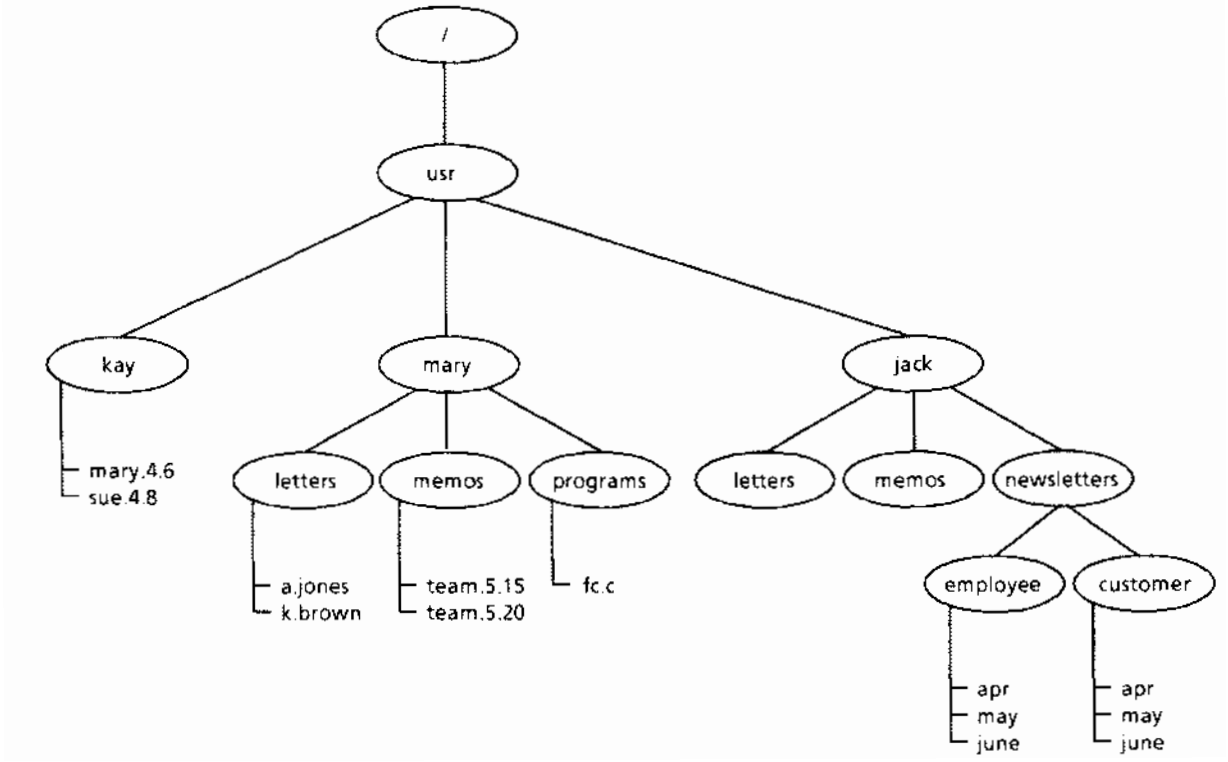
Figure 2-6. Sample Subtree

The Parent Directory

The directory immediately above another directory is called its parent. For example, in Figure 2-6, "jack" is the parent to the "newsletters" directory, and the "newsletters" directory is the parent to the "employee" and "customer" directories. A shorthand name for the parent of a directory is "..", which is pronounced "dot dot".

The /usr Directory

As you have seen, each user is allowed to organize files into meaningful hierarchies. These user hierarchies, in turn, are part of a bigger hierarchy that begins with the root directory (which is represented by **/**). Under root are several directories, including one called **/usr**. The **/usr** directory is traditionally the parent of all user directories. Figure 2-7 shows a sample hierarchy for the **/usr** directory.



F-0310

Figure 2-7. Sample **/usr** Directory with Subdirectories

Full Path Names

Each file in the system has a path name that begins at root and goes through the full path of directories to the file itself. For example, the full path name of Mary's letter to A. Jones is

```
/usr/mary/letters/a.jones
```

Slashes separate the directories when you give a full path name.

Notice that both Mary and Jack have directories with the names "memos" and "letters". In XENIX that is acceptable because each file has a different path name. Within a directory, each name has to be unique. For example, Mary can have only one "a.jones" file in her "letters" directory.

Relative Path Names

A full path name may seem like a lot to type. Fortunately, you usually do not have to type the full path name. You can start where you are and give the remainder of the path, which is called the relative path name. Figure 2-8 illustrates full and relative path names.

Full path name from root (/):	/usr/mary/letters/a.jones
Relative path name from /:	usr/mary/letters/a.jones
Relative path name from /usr:	mary/letters/a.jones
Relative path name from /usr/mary:	letters/a.jones
Relative path name from /usr/mary/letters:	a.jones

Figure 2-8. Sample Path Names

Moving from Directory to Directory

When you log in, you begin in your login directory. If you want to work on a file, you have these choices:

- Give the full path name from root.
- Give the relative path name.
- Use the **cd** (change directory) command to go to the directory that has the file, then give the file name.

For example, when Mary logs in, she is in her "mary" login directory. If she wants to go to her "letters" directory and use the **ed** line editor to edit the "a.jones" file, she can use the commands in Figure 2-9.

$ ed /usr/mary/letters/a.jones	Full path name
$ ed letters/a.jones	Relative path name from /usr/mary
$ cd letters **ed a.jones**	Change to letters directory Invoke **ed** to edit a.jones file

Figure 2-9. Path Names with Commands

The Working Directory

The directory you are in is called your working directory or your current directory. For example, when Mary goes to her "letters" directory to write a letter, "letters" is her working directory.

If you forget what directory you are in, you can use the **pwd** (print working directory) command. For example, if Mary uses the **pwd** command when she is in her "letters" directory, this results:

```
$ pwd
/usr/mary/letters
```

A shorthand name for the working directory is ".", which is pronounced "dot".

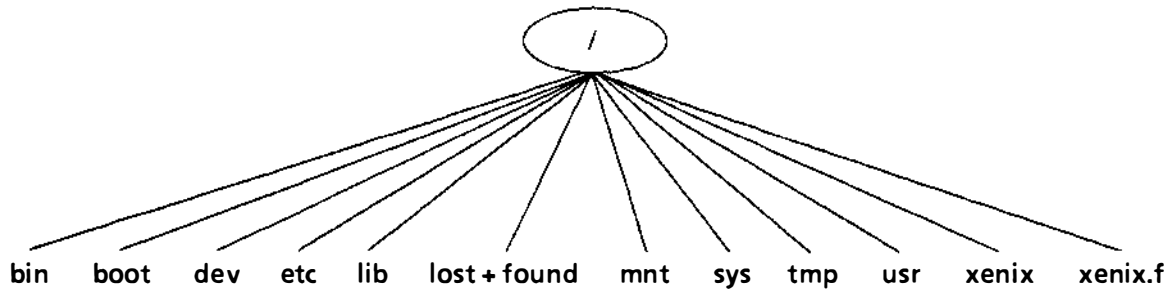
The Root Directory

Before leaving the subject of directories, you should be aware of the root directory (**/**), which has these subdirectories and files:

- **/bin** This directory has the XENIX commands that users execute most.
- **/boot** This file has the code for a program that is needed to start the system.
- **/dev** This directory contains special device files.

- **/etc** This directory has commands that are usually reserved for the system administrator plus files that the system administrator uses.
- **/lib** This directory has libraries of subroutines.
- **/lost+found** This directory lists directories that are not linked into the file system because of some problem. The entries are placed in this directory automatically by the **fsck** command that the system administrator uses regularly to check the integrity of the file system.
- **/mnt** This directory is normally used for file systems that are mounted on the root file system.
- **/sys** This directory has the code for the XENIX kernel.
- **/tmp** This directory is used for temporary files that are created by programs. These files may be removed during normal operations and they are usually removed each time the system is started.
- **/usr** This directory is used for all login directories. It is the ancestor of all user files and directories.
- **/xenix** This file has executable code for the XENIX kernel for the hard disk system.
- **/xenix.f** This file has executable code for the XENIX kernel for the flexible disk system.

Figure 2-10 illustrates the contents of the root directory.



F-0311

Figure 2-10. The Root Directory

Special Files

In a XENIX system, every hardware device is accessed using a special file. Printers, terminals, disks, tapes, and communication lines are all regarded as files. The significance of this is that you can send data to a device or read data from a device just as you would read data from an ordinary file or write data to it. For programmers, this is one of the most important features of XENIX.

Special files are contained in the **/dev** directory and only the system administrator can add special files. Device names are fixed by the system administrator, but they are treated like the names of other files. Figure 2-11 gives sample entries for the **/dev** directory. Notice that even memory is included in the list of special files, although it is rarely accessed as a file.

Device Name	Description
/dev/console	the system administrator's terminal
/dev/mem	an image of physical main memory
/dev/kmem	an image of kernel data
/dev/null	a dummy device; output sent to it is discarded
/dev/rw0a	a disk
/dev/rw0b	a disk
/dev/tty	a terminal
/dev/ttya1	a terminal
/dev/ttya2	a terminal

Figure 2-11. Sample Device Names in the **/dev** Directory

XENIX has two kinds of special files, block and character.

Block Special Files

Block special files work with one block of data (1,024 bytes) at a time. Examples are disks and tapes. A block special file may also be called a structured device. It often has a character special interface, called a raw interface, which is used by programs that perform system maintenance functions.

Character Special Files

A character special file is any special file that does not work with a block of data at a time. Examples are terminals, communication lines, printers, and main memory. A character special file may also be called an unstructured device.

File Access Permissions

In XENIX, every file belongs to an owner and a group. The owner is the person who creates the file, and the group is the group the owner belongs to when the file is created. The owner can give or deny access to anyone except the system administrator, who has access to all files on the system. The owner or the system administrator can assign a file to a new owner.

Access Permissions for Ordinary Files

For each file that you create, you can give or deny read, write, and execute permission for three different categories of users: yourself, other members of your group, and all others.

Read Permission for Ordinary Files

Reading a file means looking at its contents. Displaying a file on the terminal, printing it, compiling it, and copying it are all examples of reading a file.

Write Permission for Ordinary Files

Writing a file means changing it in some way. Adding and changing information are examples of writing a file.

Execute Permission for Ordinary Files

Executing a file means running it as a program. Most executable files are compiled programs, and you need execute permission to run them. Some executable programs are shell scripts (programs using XENIX commands and the shell programming language). You need read permission to execute a shell script. If you also have execute permission, you can execute it with the program name. For example, you can run a shell script named "check" with this command if you have read and execute permission:

```
$ check
```

If you have read permission, but not execute permission, you can run a shell script with the **sh** command. For example:

```
$ sh check
```

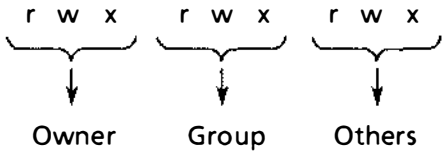
Set UID and GID

As a user, you have a user ID number (UID) and a group ID number (GID). Whenever you try to use a file, your IDs and permissions are checked. Occasionally you need someone else's ID to use a file. For example, you need root's ID to change your password in the **/etc/passwd** file, because only root can change that file. Set UID permission on the **passwd** command gives you root's ID when you use the command. Any executable file, except a shell script, can set the UID or the GID so that anyone who executes the file has the effective ID of the owner or the group owner.

Representing Permissions

For each file that you create, you can give or deny read, write, and execute permission for three different categories of users: yourself, other members of your group, and all others. These permissions may be referred to as the file mode, protection bits, or permission bits.

Permissions can be represented in two different ways. One way is to show them with characters: r for read permission, w for write permission, x for execute permission, s for set UID or GID permission, and a dash (-) for permission denied. These permissions are shown for the owner, other members of the group, and all others. For example, read, write, and execute permission for the owner, other members of the group, and all others, are represented in Figure 2-12. Permissions are often called the file mode. Examples of permissions are shown in Figure 2-13.



F-0312

Figure 2-12. Representing Permissions with Characters

File Mode	Meaning
r-x--x--x	read and execute permission for the owner execute permission for the group execute permission for others
rwxrwxr-x	read, write, and execute permission for the owner read, write, and execute permission for the group read and execute permission for others
rwxr-x---	read, write, and execute permission for the owner read and execute permission for the group no permission for others
rwsr-sr-x	read, write, and execute permission for the owner owner's permissions for anyone executing the file read permission for the group group's permissions for anyone executing the file read and execute permission for others

Figure 2-13. Sample Permissions for Ordinary Files

Permissions can also be represented with these octal numbers:

4 = read
2 = write
1 = execute
0 = deny permission

Octal numbers are part of a number system whose base is 8, just as decimal numbers are part of a number system whose base is 10. You add these octal numbers for permissions. The total is 7 for read, write, and execute permission. The total is 5 for read and execute permission. The owner, others in the group, and all others have separate totals. For example, 777 means full permissions for the owner, others in the group, and all others.

If the UID or GID permission is set, a fourth digit precedes the series. It has one of these meanings:

4 = set UID permission
2 = set GID permission
6 = set UID and GID permission

For example, 6711 gives UID and GID permission to anyone executing the file, gives full permission to the owner, and denies permission to others in the group and all others.

Figure 2-14 shows how the two different methods represent the same permissions.

Characters	Numbers	Meaning
rwxr-xr-x	755	read, write, and execute permission for the owner read and execute permission for the group read and execute permission for others
rwxr-x---	750	read, write, and execute permission for the owner read and execute permission for the group no permission for others
r-x--x--x	511	read and execute permission for the owner execute permission for the group execute permission for others
rw-rw-rw-	666	read and write permission for the owner read and write permission for the group read and write permission for others
rwsr-x---	4750	read, write, and execute permission for the owner owner's permission for anyone executing the file read and execute permission for the group no permission for others

Figure 2-14. Representing Permissions

Default Permissions

When you first receive your system, these permissions are defined for all ordinary files created in the **/usr** directory:

```
rwxr-xr-x
```

These permissions give read, write, and execute permission to the owner, read and execute permission to the group, and read and execute permission to all others. They are called default permissions because they will be assigned automatically each time you create a file.

The defaults are set with the **umask** command, which you use to define which permissions are to be removed from a base. The typical base is full permission for the owner, others in the group, and all others. This is represented in octal numbers as 777, and you subtract from 777 to get the appropriate defaults. For example, the original default removes write permission for the group and others because this command is in the **/etc/profile** file:

```
$ umask 022
```

The resulting octal number is 755 (777-022). It is often desirable to change the defaults. For example, the system administrator may change the default so files are created with full permission for the owner, read and execute permission for others in the group, and no permission for others. The octal representation for these permissions is 750 and the default is created with this command:

```
$ umask 027
```

The system administrator may place the **umask** command in the **/etc/profile** file, or you may place it in your own **.profile** file.

You can change permissions on your existing files with the **chmod** (change mode) command. For example, when Mary creates her "a.jones" letter, it has the default permissions. If she wants to include write permission for others in her group, she can use this command:

```
$ chmod g + w a.jones
```

Access Permissions for Directories

Directories can be read, written, or searched.

Read Permission for Directories

Reading a directory means looking at the contents of the directory file itself. Since a directory contains only a list of file names and their inode numbers, reading it means using the **ls** command to look at the list. Figure 2-15 shows what kind of information is available to you if you have read permission on a directory. The **-i** option of the **ls** command shows inode numbers and file names.

```
$ ls -i /usr/mary
450 letters
460 memos
475 programs
$
```

Figure 2-15. Reading a Directory

Keep these rules in mind:

- Read permission on a directory does not give you access to the contents of the files in the directory. You can only read the names of the files.
- If you know a file's name and have read permission on it, you can see its contents provided you have search permission on its directory. You do not need read permission on its directory.

Write Permission for Directories

Writing to a directory means creating a new file (including a subdirectory) in it or deleting a file from it. It may help to picture the actual contents of the directory file. Writing to a directory means adding a name to the list of files or removing a name from the list.

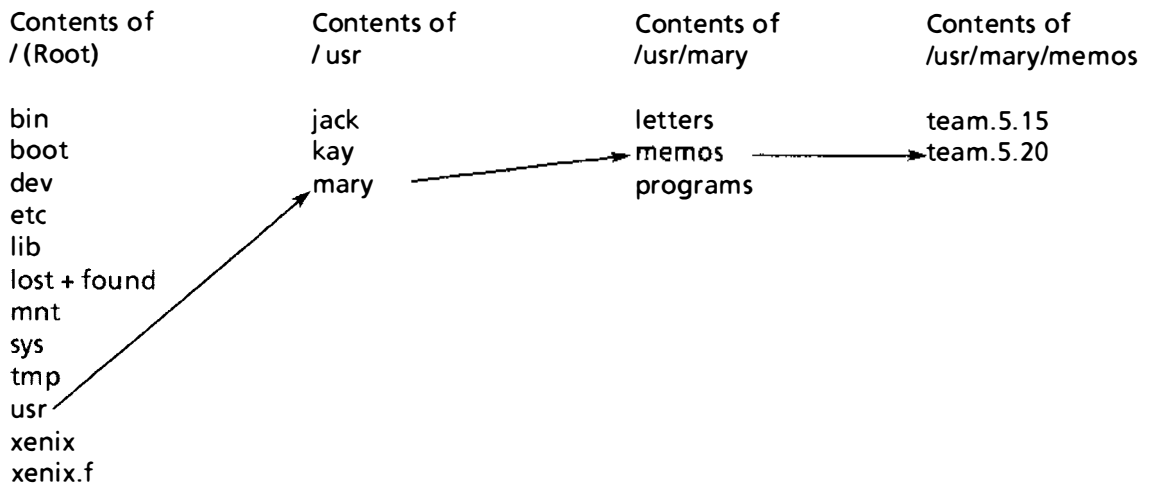
Keep these rules in mind:

- You do not need write permission on a file to delete it. You just need write permission on the directory. You will be warned if you try to delete a file without having write permission, but you can still delete it.
- You can change the contents of a file if you have write permission on the file. You do not need write permission on the directory.

Search Permission for Directories

Directories have search permission instead of execute permission. It is meaningless to execute a directory, since it is not a program. The "x" is still used as the symbol for search permission.

Searching a directory means going to the directory with the **cd** (change directory) command or searching through its list of files when a file name is given. You cannot use a file name successfully unless you have search permission for every directory in the path. Figure 2-16 shows how directories are searched when the full path name is "/usr/mary/memos/team.5.20".



F-0313

Figure 2-16. Searching Directories

Access Permissions for Special Files

System owners, such as **root** and **bin,** own all of the special files. Others usually have write permission for terminals and printers and no permission for other devices.

Special files for terminals are usually owned by root when they are not being used. When you log in, you become the owner temporarily and can set the access permissions on the terminal. When you log off, ownership reverts to root.

Trying to execute a special file is meaningless.

Read Permission for Special Files

Reading a special file means looking at its contents. For example, if you deny others read permission for your terminal, they cannot read what you are typing. Read permission for a printer is meaningless.

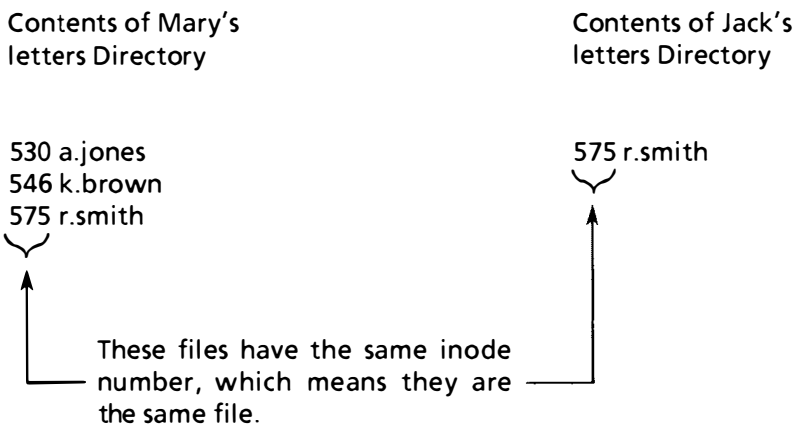
Write Permission for Special Files

Writing a special file means sending data to it. For example, if you give others write permission on your terminal, they can send messages to your screen. People usually give others write permission on their terminals by using the **mesg** command to permit or prevent messages from reaching the terminal. The **mesg y** command permits others to send messages to your terminal and the **mesg n** command prevents others from sending messages to your terminal.

Links to Files

A file exists somewhere on a disk and you use its name to work with it. This name is for your convenience and it is stored in the directory, not in the file itself. XENIX knows each file by a unique number called an inode number. These facts make it possible for you to have names for a file in more than one directory and to give a file more than one name. This is called linking.

Imagine that Mary and Jack are writing a joint letter to R. Smith. They both want to list the file in their own directories so that they can use the short, relative path names. They accomplish this by having Mary create the file and having Jack use the **ln** command to create a link to it. He can use the same name for the file or use a different name. Figure 2-17 illustrates links.



F-0314

Figure 2-17. Links to a File

These rules apply to links:

- You need search permission on a directory to link to files named in it.
- No one can link across file systems.
- All directories have at least two links, because they have the shortcut name "." in addition to their full name.
- If you delete a linked file from your directory, the file itself is not deleted unless no links remain.

Working with Files

XENIX offers many commands for working with files. For example, XENIX gives you commands to

- Create files
- Edit files
- Compare files
- Identify file types
- Display files
- Display the first few or last few lines of a file
- Display a file one screen at a time
- Divide files
- Join files
- Sort files
- Copy files
- Encrypt files
- Rename files
- Delete files
- Move files to another directory
- Count the characters, words, and lines in a file

Appendix A includes a summary of the Basic System's commands for working with files.

Logical Files and Physical Locations

As a user, you work only with files and directories. You do not have to be concerned with finding a place for them on the disk or locating them after they have been stored. Those are problems that the kernel solves and it solves them by implementing a file system.

A disk is a mass storage device that holds millions of characters called bytes. A file system is a physical partition of a disk. It treats the physical area of a disk as a series of blocks (each block equals 1,024 bytes) and imposes a logical organization upon them. A file system stores data as efficiently as it can, then finds it as quickly as possible when you want to use it.

Logical Files

Logically, a file is a series of bytes, as illustrated in Figure 2-18.

MEMO

TO Team
FROM Mary
DATE May 15
SUBJECT Revised Schedules

Please give me your revised schedules by Friday.

Figure 2-18. Sample File

Notice that the file has nothing but the text of a memo. It does not even have the file name. You see it here as one continuous series of characters, but parts of a large file are in different blocks on the disk.

It is the file system that makes the connection between the logical file that you create and the physical blocks that it occupies on the disk.

Finding the Physical Location of File Data

The file system keeps several pieces of information that the kernel needs to find a file on the disk:

- The name. You identify a file by name when you want to use it.
- The directory. When you give a file name, the kernel searches the directories listed in your search path until it finds the file name. Along with the file name, the directory has the inode number that the kernel has assigned to identify the file.
- The inode list. The kernel uses the inode number in the directory to find the inode number in the inode list, which has all of the inode numbers in the file system. For each inode, an index entry gives this information about the file:
    - File type. This identifies the file as an ordinary file, a directory, a special file, a semaphore, or a named pipe. (Semaphores and named pipes are discussed in the *XENIX 286 C Library Guide*.)
    - Permissions. This identifies read, write, execute, and set UID and GID permissions.
    - Owner. This gives the UID of the owner of the file.
    - Group. This gives the GID of the group the file belongs to.
    - Number of links. This identifies the number of times the file is listed in directories.
    - File size in bytes.
    - Date the file was created.
    - Date the file was last read.
    - Date the file was last modified.
    - Location of the file on the disk. This entry lists up to 13 blocks. The first ten blocks of the file are listed here. Three additional entries give the addresses of the blocks that tell where the rest of the file is located. For example, if the file has more than ten blocks, an entry points to a block that lists the next 128 blocks of the file. These blocks that point to other blocks are called indirect blocks.

Figure 2-19 illustrates how the kernel uses the file name, directory, and inode list to find file data when Mary goes into her "memos" directory and asks to print "team.5.15". The kernel finds the name of the file in the directory, uses the inode number to find the file on the inode list, then uses the location in the inode list to find the file on the disk.

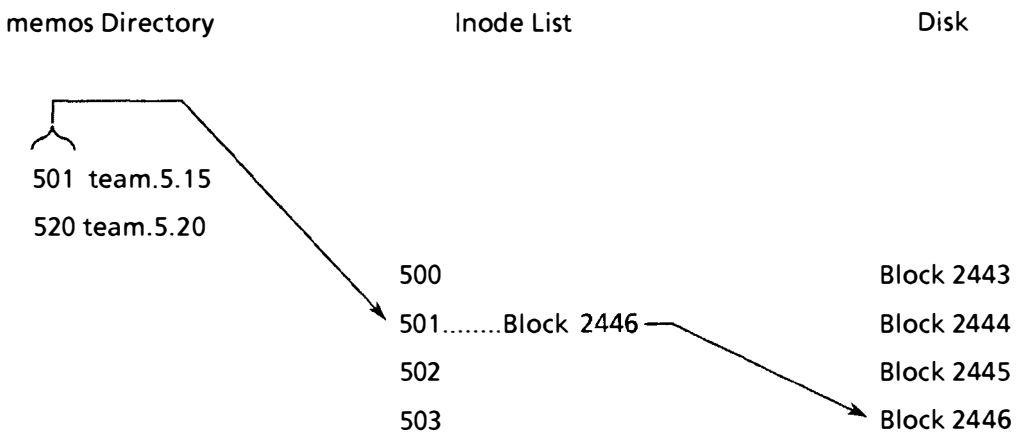


Figure 2-19. Logical Files and Physical Locations

The Structure of a File System

A file system is a partition of a disk. Each Winchester disk sold by Intel has one root file system. If it has 20 megabytes or more, it also has a separate user file system for all user files. Very large disks may have even more file systems. If the disk has fewer than 20 megabytes, there is only one file system, the root file system, and the user files are part of it.

Figure 2-20 illustrates the root file system and user file system as partitions of a 40-megabyte Winchester disk.

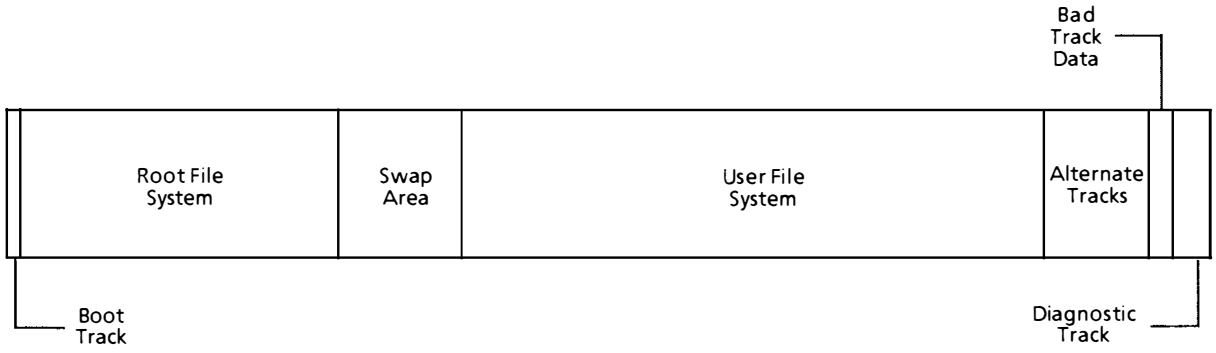
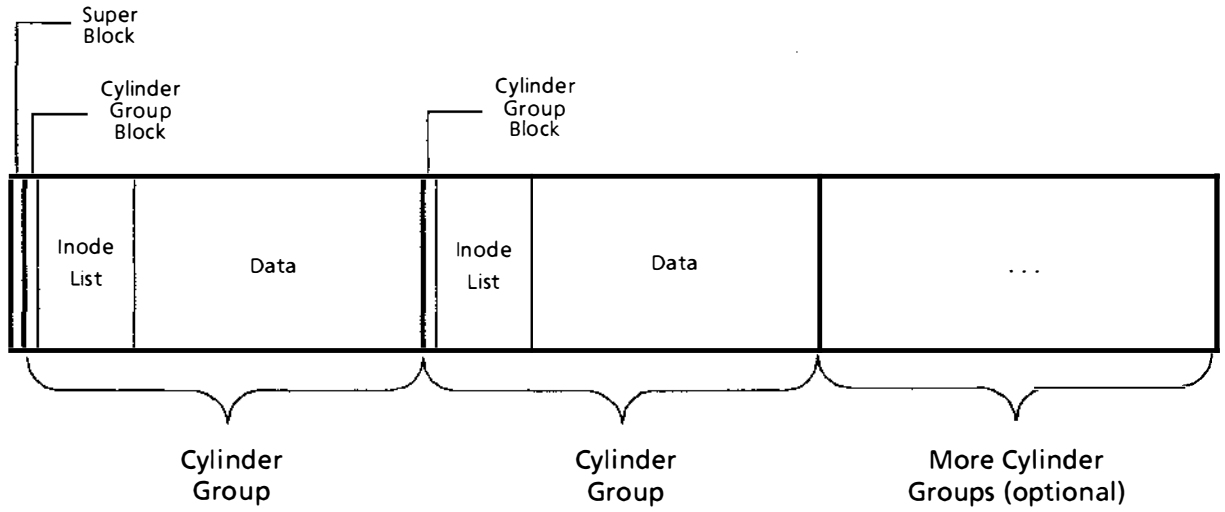


Figure 2-20. The Structure of a 40-Megabyte Winchester Disk

Boot Track	The boot track has a program that loads the XENIX kernel into memory when the system administrator starts the computer.
Root File System	The root file system is the first file system on the disk.
Swap Area	The swap area always follows the root file system. It is the area where processes can be placed while they wait for their turn to execute.
User File System	A disk with more than 20 megabytes has a separate user file system.
Alternate Tracks	If a regular disk track is bad, an alternate is assigned.
Bad Track Data	If testing shows that a track is marginal, it is listed here as a bad track. During disk formatting, the bad track information is read and alternate tracks are assigned.
Diagnostic Track	This is used for hardware diagnostics.

A file system is made up of a super block and a series of cylinder groups. These cylinder groups are made up of a cylinder group block, inodes, and data. The number of cylinder groups depends on the size of the disk and the needs of the installation. You may check the *XENIX 286 Installation and Configuration Guide* for details.

Figure 2-21 illustrates the structure of a file system.



F-0317

Figure 2-21. The Structure of a File System

Super Block	The first block of a file system is the super block. It gives the location of each cylinder group.
Cylinder Group Block	The cylinder group block gives the location of the inode blocks and data blocks in the group. It includes a bit map that shows which data blocks have been allocated and which ones have not.
Inode List	The inode list has information for each file.
Data	The data area is used for files.

Cylinder Groups

Cylinder groups have been introduced to enhance the performance of the operating system. How they work is not discussed in detail here because normally only the person who installs the system is aware of them. One of XENIX's assets is that most users do not have to be aware of the physical organization of data.

What is important to most users is that the kernel spends less time looking for file data when the file system is divided into cylinder groups. This is because

- The inodes are closer to the data, so it takes less time to access data after the inode is located.
- As much as possible, contiguous blocks are used for files. It is faster to access blocks that are together than blocks that are scattered over the disk.

File Allocation

Each cylinder group block has a bit map that shows whether the data blocks in the cylinder group have been allocated to a file. The map is a series of bits, one for each block. If the block has been allocated, the bit is set to 0. If the block is free, the bit is set to 1.

In earlier versions of the file system, a free list of blocks was used instead of a bit map. When a block was needed, the first block on the list was allocated. When a block was no longer used, it went to the top of the free list. The result was that the list of free blocks became random eventually. It was unlikely that files would have contiguous blocks because only one block was allocated at a time and the blocks were not in order on the free list.

With the bit map, blocks are always listed in order and the kernel can more often find contiguous blocks for a file. This leads to more consistent performance over time.

The Root File System and the Root Directory

The root file system is not the same as the root directory. The root file system is a physical partition of the disk. It is created by the system administrator during installation, and it usually includes all of the system directories.

The root directory is the parent of all files. This means that even files in other file systems have to have a path back to root to be used. For example, if you have a 40-megabyte Winchester disk, you have two file systems, root and user, which occupy two separate partitions of the disk. The user file system has to be attached to some empty directory on root's hierarchy of directories before you can work with files in its file system. Attaching file systems is called mounting them and it is normally done when the system administrator starts the system.

intel®

CHAPTER 3
RUNNING PROGRAMS

Programs and Processes

Programs and processes are two important concepts in XENIX. You use the computer to run programs, and the computer runs them by starting processes. Running a program is a matter of starting a process, but a process and a program are not the same. For example, if four users execute the **ls** command, only one copy of the program is used, but four different processes begin.

Programs

Programs are instructions that perform some function. They fall into several categories:

- XENIX commands. Most XENIX commands are executable programs.
- Shell programs. A shell is a XENIX command interpreter. It is also a programming language with variables, arguments, conditional statements, case statements, for statements, while statements, and comments. You may write shell programs that use both XENIX commands and features of the shell programming language. These programs are called shell scripts.
- Source programs. A source program is a set of instructions that someone has written in a high-level language such as C.
- Object programs. An object program is a source program that has been compiled and is ready to be executed.

Executable programs are usually stored in the **/bin** directory, the **/usr/bin** directory, the **/etc** directory, or a user's directory. Several different users may execute the same program at the same time. For efficiency, only one copy of the program is brought into memory to be executed.

Processes

Processes are programs being executed. Each time a program is executed, a process begins. It is unique and is identified by a number called a PID (process ID). Like directories, processes are organized into hierarchies. The first process (PID 1) begins when the system administrator starts the system, and all processes descend from process 1 in parent-child relationships. For example, when process 1 starts process 2, process 1 is the parent and process 2 is the child.

A process begins as a result of an **exec** system call or a **fork** system call. An **exec** replaces another process and takes its PID. It is used when one process is finished and will not be needed again. A **fork** starts a child process and continues to let the parent process run. The child process inherits all of the open files of the parent but is separate from the parent and has its own PID. The parent process either waits until the child process ends, or the parent continues to run while the child is running.

Figure 3-1 illustrates the hierarchical structure of processes and the difference between an **exec** and a **fork.**

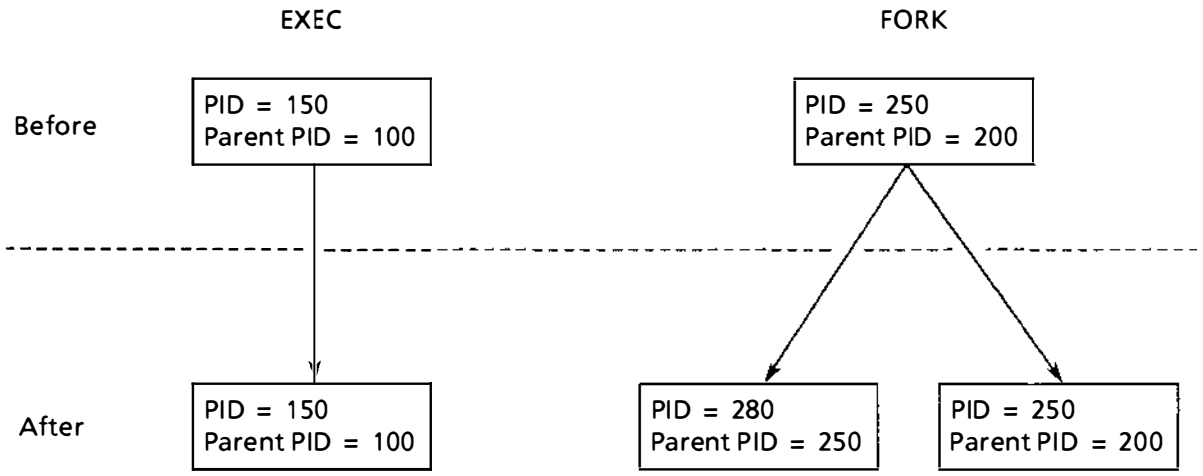


Figure 3-1. Creating a New Process with an **exec** or a **fork**

The kernel keeps track of all processes in a process table and a user table. The number of processes that can run at one time depends on the size of the tables, and the size is defined by the system administrator during configuration.

What Happens During System Startup

The system administrator starts the system by turning on the hardware and loading the XENIX kernel. The kernel starts a program called **init**, which runs as process 1. Process 1 is at the top of the process hierarchy and it runs as long as the system is up. All processes are its descendants.

The system administrator usually brings the system up in single-user mode, does system maintenance, then puts the system in multiuser mode. Several processes need to be started before users are allowed on the system.

First, a shell script called **/etc/rc** is executed. This script contains commands to

- Mount file systems on the root directory tree.
- Clear temporary files.
- Start daemons. (Daemons are programs that run continuously. For example, **lpd** is a daemon for the line printer. It is always ready for a print command. Another daemon, **cron**, checks the commands in the **/etc/crontab** file and executes them at the assigned time.)

Next, the kernel checks the **/etc/ttys** file. This file has a list of terminals with these seven characters that describe each terminal:

- One character that tells whether the terminal is enabled (1 for enabled, 0 for disabled).
- One character that gives the terminal's characteristics to a program called **getty**.
- Five characters that give the terminal name in the **/dev** directory (for example, ttya1, ttya2).

A **getty** process is started for each enabled terminal. Once **getty** has initialized the terminal characteristics and determined the correct baud rate (the rate at which characters are transmitted), the **getty** process replaces itself with a **login** process. This is an example of an **exec**. The **getty** process is no longer needed, so it is replaced by the **login** process. This means that if the **getty** process had PID 5 in the process table, the **login** process takes its place and has PID 5.

How You Gain Access to the System

You cannot do anything until the system administrator has created an account for you on the system. The system administrator uses a command called **mkuser,** which prompts for information about the account, places information in the **etc/passwd** file, and creates files that you will need, such as a mail box for electronic mail and a **.profile** file that has information about you and your terminal.

The /etc/passwd File

The **/etc/passwd** file controls the login procedure that you must complete to gain access to the computer. The **/etc/passwd** file has this information:

- A login name. This is the name that you will type when you want to log on the system. On many systems, it is your first name in lowercase letters.
- Your password in encrypted form. The system administrator assigns a password when you are added to the system, but you may change it at any time with the **passwd** command.
- A unique user ID number (UID). This number identifies you in the system. UIDs for regular users start with 200. Number 0 is reserved for root. Numbers 1-199 are reserved for special "users" who own system files. Examples are **bin** and **cron.**
- A group ID number (GID). If people at your installation need to share certain files, the system administrator may define groups. You can be a member of several groups, but you can work in only one group at a time. This number identifies the group you are in when you log on.
- A comment that can be used for reference information. The **finger** command that displays information about users expects this field to have a user's full name, office, phone extension, and home phone number. It is not necessary to include all of these pieces of information, but if you do, separate them with commas, as shown in Figure 3-2. Another name for this field is GCOS. (The initials have historical significance only.)
- The name of your login directory. The login directory becomes your working directory immediately after login.
- Your login shell. Your login shell is the command interpreter you use. It can be the Bourne shell, the C shell, the restricted shell, or the visual shell. It can even be a specific program. For example, if you use the system only for word processing, the word processing program can be listed as your login shell so you will go into it immediately after login. If no shell is specified, the Bourne shell is used.

Each piece of information in the **/etc/passwd** file is separated by a colon. Figure 3-2 shows a sample entry from the file.

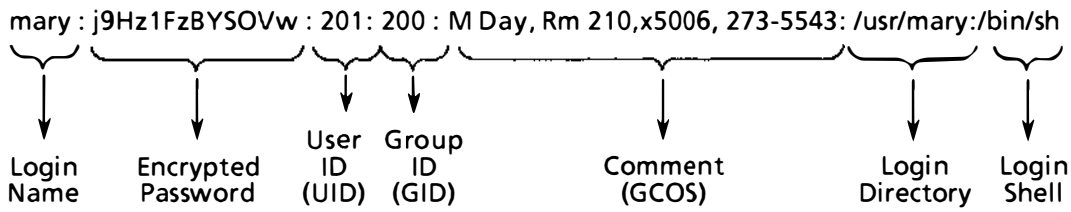


Figure 3-2. Sample Entry in the **/etc/passwd** File

What Happens During Login

Logging in is the procedure that you follow to gain access to the computer. It involves typing a login name and giving a password.

You can log in when you see this prompt on the screen:

```
login:
```

You type the login name that the system administrator has given you. The screen displays a prompt for the password, and you have about one minute to give your password. The **login** process checks the **/etc/passwd** file for your login name, encrypts the password you typed, and compares it to the encrypted password in the file.

If there is mail in your mail box, you are notified that you have mail when you log in.

The Login Shell

If your login name and password are valid, the **login** process moves you to your login directory and uses an exec to start your login shell. Since an **exec** is used, the shell has the same PID that your **login** process had.

At the same time, the name of the login directory is stored in a variable called **HOME** and the login name is stored in a variable called **LOGNAME.** Variables have values that vary from user to user. Some variable names, like **HOME,** are predefined and are always entirely in capital letters. Later in this chapter, some variables will be discussed in more detail.

The login shell continues to run until you log off by pressing the CONTROL key and the D key at the same time. This key combination is referred to as CONTROL-D.

The following discussion of the login shell is based upon the Bourne shell, which is referred to simply as the shell. The C shell, restricted shell, and visual shell differ in some ways and are described later in this chapter.

The Standard Input, Output, and Error Files

When the login shell is started, several things happen internally. The terminal is opened as the standard input file, the standard output file, and the standard error file. This means that all input will come from the terminal, all output will be displayed on the terminal, and all error messages will be displayed on the terminal unless you specifically open other files for them. In XENIX, each open file has a number, called a file descriptor, associated with it. The standard input is opened with file descriptor 0, the standard output is opened with file descriptor 1, and the standard error is opened with file descriptor 2.

Default Variables Set by the Login Shell

The shell is a program that you use to execute commands. It stores several pieces of information that it needs in variables. The **HOME** variable, for example is defined at login so that the shell will know your login directory. Other variables are given default values when the login shell starts. A variable is always defined by giving the name of the variable, an equal sign, and the value of the variable.

The variables set by the login shell and their values are defined below:

- **PATH.** When you give a command, the shell searches through the directories named by the **PATH** variable until it finds the program to be executed. You define a variable by giving its name, an equal sign, and its value. The default search path is

  ```
  PATH = /bin:/usr/bin:$HOME/bin:.
  ```

  The directories are separated by colons, so the search path is through the **/bin** directory, then the **/usr/bin** directory, then the **/bin** directory in your working directory (this **/bin** directory is optional), then your working directory (.). Programs for commands that most users can execute are usually stored in one of these directories.

- **PS1.** **PS1** stands for prompt string 1, which is the main prompt that the shell displays when it is ready to accept commands. The default is

  ```
  PS1 = "$ "
  ```

  The prompt is shown in quotes here because it includes a space. When the prompt itself appears on the screen, it is a dollar sign followed by a blank space. If you want some other prompt, you redefine the variable. For example, if you want to be prompted with "Ready ", you use this definition:

  ```
  PS1 = "Ready "
  ```

- **PS2.** **PS2** stands for prompt string 2, which is the prompt that the shell displays if you need to give more information. The default is

  ```
  PS2 = "> "
  ```

- **IFS.** IFS stands for internal field separators, which the shell recognizes as characters that separate fields. The defaults are a space, a tab, and a newline character. You should use this variable only if you are doing shell programming.

The .profile Files

After the shell has set default variables, it reads the **/etc/profile** file, which has information that applies to all users, then reads the **.profile** file in your login directory, which has information that applies only to you. The information in your **.profile** file is usually a combination of commands and definitions of variables that your login shell needs each time it starts. It is placed in your login directory when the system administrator adds you to the system, and you may change the information in the file at any time. Sample entries in the **.profile** file are shown in Figure 3-3.

```
PATH = /bin:/usr/bin:$HOME/bin:.
TERMCAP = /etc/termcap
TERM = h8020e
MAIL = /usr/spool/mail/$LOGNAME
export TERMCAP TERM PATH MAIL
```

Figure 3-3. Sample **.profile** File

In Figure 3-3, the **PATH** variable is being redefined. The new value replaces the default value that the shell had set. Notice the **$HOME/bin** directory in the path. When the name of a variable begins with a dollar sign, it means to use the value of the variable. **$HOME** means to use the value of the **HOME** variable. Imagine that

```
HOME = /usr/mary
```

In this case, the new search path is the **/bin** directory, the **/usr/bin** directory, the "/usr/mary/bin" directory, and the working directory. The dot at the end of the path is the shorthand name for the working directory.

Three other variables are usually defined in your **.profile** file:

- **TERMCAP** identifies the file that has descriptions of terminals. The default is **/etc/termcap**, and the variable is rarely redefined.
- **TERM** identifies the terminal by a short code name. For example, h8020e is the name for the Hazeltine Executive model 20.
- **MAIL** identifies the file that keeps your mail. When **$LOGNAME** is used as the last part of the path name, it means to use the value of the **LOGNAME** variable. If the **LOGNAME** were "mary", the mail box would be the "/usr/spool/mail/mary" file.

You use the **.profile** file for commands that you want to execute at login as well as variables that you want to define. The most common command in this file is the **export** command. It is included so that the variables that are named will be defined in any new shells that the login shell starts. Without the **export** command, the variables would be defined only in the login shell.

Executing Commands with the Shell

When the shell is ready for commands, the shell prompt appears on the screen. This is a dollar sign unless you have changed the **PS1** variable. When you give commands, the shell interprets them and forks a new process to execute each one. The general term for a program that does these things is a command interpreter. Some operating systems have only one command interpreter because it is in the kernel. XENIX has several different command interpreters to provide maximum flexibility. Each one is a C program that can be changed or replaced with another C program.

Executing Simple Commands

At the shell prompt, you can type a command and press the RETURN key. For example, if you want to see who is on the system, you can use this command:

```
$ who
```

This sequence of events follows:

- The shell interprets the command line.
- The shell searches for an executable program with the same name as the command. It looks in each directory listed in the search path defined by the **PATH** variable. Imagine that this is the search path:

  ```
  /bin:/usr/bin:mary/bin:.
  ```

  The shell searches the **/bin** directory for the **who** program and finds it there.
- The shell forks a child process for the **who** process and waits.
- The child attempts to **exec** (load) the **/bin/who** program.
- The kernel finds these permissions on the **/bin/who** file:

  ```
  rwx--x--x
  ```

  Mary belongs to the category of others, so she has execute permission.
- The kernel executes the **/bin/who** program and the output is displayed on the terminal. This sample display lists the users who are logged on and identifies their terminals and login times:

  ```
  mary    ttya1    Jul 12 10:15
  jack    ttya2    Jul 12 11:03
  ```
- The kernel signals to the shell that the child process has finished executing the **/bin/who** program.
- The shell wakes up and prompts for the next command.

Using Options

Many commands have options. For example, the **ls** command can be used with or without options. If you use it as a simple command, the contents of your working directory are displayed. For example, **ls** would produce this alphabetical list of files in Mary's "memos" directory:

```
$ ls
team.5.15
team.5.20
$
```

The **ls** command has several options. The **-l** option, for example, gives this information about files: the permissions, number of links, owner, group, size in bytes, and time of last modification. When you use options, you give them after the command name. For example:

```
$ ls -l
total 2
-rw-r--r--   1    mary      200      59 May 15 10:15 team.5.15
-rw-r--r--   1    mary      200      30 May 20 10:15 team.5.20
$
```

The total refers to the number of blocks.

Another option, **-s**, shows the number of blocks for each file:

```
$ ls -s
1 team.5.15
1 team.5.20
$
```

When a command has several options, you can often use more than one at a time. For example, this command uses two options, **-l** and **-s**:

```
$ ls -ls
total 2
1 -rw-r--r--   1    mary      200      59 May 15 10:15 team.5.15
1 -rw-r--r--   1    mary      200      30 May 20 10:15 team.5.20
$
```

Using Arguments

With some commands, you name the files or directories to be used. These files or directories are called arguments to the command and they appear on the command line after any options.

This is an example of the **ls** command with the "memos" directory as an argument:

```
$ ls memos
team.5.15
team.5.20
$
```

This is an example of the **ls** command with an option and an argument:

```
$ ls -s memos
total 2
1 team.5.15
1 team.5.20
$
```

Using Metacharacters

Before the shell sends commands, options, and arguments to a program to be executed, it interprets them, paying special attention to special characters called metacharacters or wildcards. These characters are described in Figure 3-4.

?	Matches any one character
*	Matches any string of characters
[ ]	Defines a set of characters
-	Defines a range of characters within a set
!	Negates a set of characters

Figure 3-4. Shell Metacharacters

The shell interprets these characters, generates complete file names, and sorts them alphabetically before it sends the arguments to the program being executed. The significance of this is that you can give files names that will let you take advantage of metacharacters.

The ? Metacharacter

The ? metacharacter matches any one character. For example, suppose that you are writing a book with five chapters. If you follow a pattern in naming files, such as "Chap1", "Chap2", "Chap3", "Chap4", and "Chap5", you can use the ? metacharacter when you want to print all five chapters:

```
$ lpr chap?
```

The shell interprets the command and generates complete file names before sending the arguments to the program, so the program never sees the metacharacters. It always receives complete arguments. In this example, the shell generates these file names and sends them to the **lpr** program:

```
chap1 chap2 chap3 chap4 chap5
```

The * Metacharacter

The * metacharacter matches any string of characters. For example, this command displays the contents of all of the files whose names begin with "memo" and end with any series of characters:

```
$ cat memo*
```

The [and] Metacharacters

The [and] metacharacters define a set of characters. For example, this command prints "chap1", "chap4", and "chap5":

```
$ lpr chap[145]
```

The command does not print "chap2" or "chap3" because they are not identified in the set.

The - Metacharacter

The - metacharacter defines a range of characters. For example, this command prints "chap1", "chap2", "chap3", and "chap4":

```
$ lpr chap [1-4]
```

The ! Metacharacter

The ! metacharacter defines the characters that are not included in a set. For example, you can use this command to print all chapters except 1-4:

```
$ lpr chap [!1-4]
```

Redirecting Input and Output

All of the programs that you run assume that the input is coming from the standard input and that the output is going to the standard output, so they do not have to be concerned with input and output devices. If you want to take input from some source other than the terminal or send it to some other destination, you can have the shell redirect input or output.

For example, if you use the **ls** command to print a list of files, the list appears on the terminal. If you want to place the list in a file, you use an output redirection symbol (>) to have the shell redirect it. This command places the list in a file called "list":

```
$ ls >list
```

If the file does not exist, it is created. If the file does exist, the new contents overwrite it unless you use >> to add to the end of the file instead. For example, this command adds the output of the **ls** command to the end of the "list" file:

```
$ ls >>list
```

Input can also be redirected. The shell expects input to come from the terminal, but you can use the input redirection symbol (<) to bring input from some other source. For example, when you use the **mail** command, the input (message) normally comes from the terminal. If you want to send a message to Jack, for example, you use the **mail** command with Jack's login name as an argument, then begin typing the message on the next line. After you have completed the message, you go to a new line and press CONTROL-D. This is an example:

```
$ mail jack
Please send your draft proposal to Mark.
CONTROL-D
```

The **mail** command also takes input from a file if you use input redirection. For example, Jack can send his "proposal" file to Mark by giving this command:

```
$ mail mark < proposal
```

Pipes

You often need to perform more than one operation on data. For example, you may want to get data, then sort it. You can do this most efficiently with a pipe, which is a tool that connects the standard output of one command to the standard input of another command. The symbol for a pipe is |.

Suppose that you want an alphabetical list of users who are on the system. The **who** command supplies a list of users who are logged on, but it lists them by terminal, beginning with the console, which is the system administrator's terminal. For example:

```
$ who
sarah       console     June 29 09:25
jack        tty1        June 29 10:15
mary        tty2        June 29 11:45
```

If you use a pipe, you can write one command line that sends the output of the **who** command to the **sort** command and displays the sorted, alphabetical list on the terminal. For example:

```
$ who | sort
jack        tty1        June 29 10:15
mary        tty2        June 29 11:45
sarah       console     June 29 09:25
```

If you want to print the alphabetical list on a printer, you can add another pipe:

```
$ who | sort | lpr
```

In this case, the list does not appear on the terminal. It goes directly to the printer.

With pipes, you need fewer command lines because you do not have to create temporary files and move data from one file to another. This series of commands illustrates the steps you would have to complete to print an alphabetical list of logins if you did not use pipes:

```
$ who >logins
$ sort logins >printlogins
$ lpr printlogins
```

The list of users logged on is redirected to the "logins" file, then the contents of the "logins" file are sorted and redirected to the "printlogins" file, then the "printlogins" file is printed. The **sort** command does not change the contents of the "logins" file itself. It just takes those contents and sorts them for the standard output. In this case, the output is redirected to another file.

Filters

Some commands take data from the standard input, use or change the data, and display the result on the standard output. These commands are called filters and they are often used with pipes.

Suppose that you want to combine and sort two lists of names and phone numbers. The easiest way is to use pipes and filters. The first list, "list1", has these lines:

```
Mary      4451
Jack      4452
Sharon    4563
Mark      5441
```

The second list, "list2", has these lines:

```
Dan       7787
Jan       7733
Kent      6765
```

The **cat** command joins files and the **sort** command sorts them. This command line combines the lines of "file1" and "file2", sorts them, and displays the output on the terminal:

```
$ cat file1 file2 | sort
Dan       7787
Jack      4452
Jan       7733
Kent      6765
Mark      5441
Mary      4451
Sharon    4563
```

The input files, "file1" and "file2", are unchanged.

Figure 3-5 lists the filters used most.

awk	change lines that match patterns
dd	convert and copy a file (to process other systems' data)
grep, egrep, fgrep	select lines that match or reject patterns
head	print the first few lines of a file
nl	add line numbers to a file
sed	edit a file according to a script of commands
sort	sort a file
tail	print the last part of a file
tr	copy and translate characters
uniq	remove repeated lines from a file
wc	count the lines, words, and characters in a file

Figure 3-5. Common Filters

XENIX Shells

This chapter has explained how the standard Bourne shell interprets your commands and passes information to the programs you want to execute. The Bourne shell is powerful and works well for many users. The Bourne shell is supplemented by the restricted shell, visual shell, and C shell for this release so you can choose the command interpreter that works best for you. Additional shells are available from other sources.

Bourne Shell

The shell that has been discussed in this chapter is the standard Bourne shell (named after its creator, S. R. Bourne). Its program name is **sh** and its standard prompt is a dollar sign ($). The Bourne shell is able to redirect input and output, interpret metacharacters, use pipes with filters, use variables, and serve as a programming language.

Restricted Shell

The restricted shell is a subset of the Bourne shell. Its program name is **rsh.** If your use of the system is limited, the system administrator may give you this shell and define the commands you can execute. The restricted shell has the features of the Bourne shell, but it does not allow you to change directories with the **cd** command, define your own search path, use any command names that have slashes (typically commands in the **/etc** directory, which are reserved for the system administrator), or redirect output.

Visual Shell

The visual shell is a menu that lists the most common commands plus the application programs your installation uses. Its program name is **vsh** and it is similar to the user interface for Microsoft's Multiplan software. The system administrator may give you a visual shell if you spend most of your time running application software.

C Shell

The C shell is a variation of the Bourne shell developed at the University of California at Berkeley. Its program name is **csh** and its standard prompt is a percent sign (%). The name is C shell because it has features in common with the C programming language. Like the Bourne shell, the C shell is able to redirect input and output, interpret metacharacters, use pipes with filters, and use variables. It also has these features:

- A history function that keeps a list of commands you have used recently (you define the number to be kept) so that you can reuse them without retyping them
- Ability to process arrays
- An alias function that you can use to change command names and create new commands

intel®

CHAPTER 4
TEXT PROCESSING

Tools for Text Processing

XENIX has a full set of tools for working with text files. This is partly because document production programs were among the first tools developed for the UNIX system and partly because programmers and writers use many of the same tools. XENIX offers assistance at each of these stages of a typical writing project: create a draft, check it, revise it, and produce a final version. You need the Extended System to check a document and format it with standard options such as centering and bolding.

Tools for Creating a Draft Document

First you type a draft document with a text editor. The **vi** editor is a popular choice because you can work with an entire screen of material at a time when you use it, but you can also use the **ed** or **ex** line editor and work with one line or a series of lines at a time.

The text you type is a series of lines without paragraph divisions, centering, or other features of a finished document. As you type the lines, or at some time before producing the final version, you put formatting instructions in the document. These instructions are codes that tell how to treat text. For example, there are codes for centering, for starting paragraphs, for bolding words, and for creating lists.

The different code types are

- **nroff/troff** codes. The term **nroff** stands for new runoff, which refers to printing on a printer, and **troff** stands for typeset runoff. Each **nroff/troff** code accomplishes one specific thing, such as justifying a line, printing a page header, printing multiple columns, numbering columns, setting the line length, or indenting a line. The **nroff** codes format text for a printer and the **troff** codes format text for a phototypesetter. The basic **nroff** and **troff** codes are the same, but **troff** has some extra options, such as proportional spacing, different fonts (including roman, italic, and bold), Greek and mathematical characters, and different type sizes.

  Each **nroff/troff** code begins with a dot and has lowercase letters. It goes on the line above the text to be formatted.

- **mm** macros. A macro represents a series of **nroff** or **troff** instructions that accomplishes some routine function such as starting a paragraph or creating a list. With the **mm** macros in the Extended System you can prepare letters, memos, and other office documents. You can also create your own macros.

- **eqn/neqn** codes. You use **eqn/neqn** codes for mathematical equations. The **eqn** program interprets the codes for a phototypesetter and the **neqn** program interprets them for a printer.

- **tbl** codes. You use **tbl** codes for tables.

The following figures illustrate **nroff/troff** codes and **mm** macros. Figure 4-1 illustrates an **nroff/troff** code and Figure 4-2 illustrates the formatted line. Figure 4-3 illustrates how **mm** macros can be used to produce a list with bullets and Figure 4-4 shows the resulting list. Notice that the macros begin with a dot and are capitalized. The **.BL** macro stands for bullets, the **.LI** macro marks each list item, and the **.LE** macro marks the end of the list.

```
.ce
This sentence will be centered.
```

Figure 4-1. Sample **nroff/troff** Code

This sentence will be centered.

Figure 4-2. Sample Formatted Line

```
.BL
.LI
This is the first item.
.LI
This is the second item.
.LE
```

Figure 4-3. Sample Use of Macros

- This is the first item.
- This is the second item.

Figure 4-4. Sample Formatted List

Tools for Checking a Draft Document

After you have created a document, you can check it with several different XENIX commands. The **spell** command, for example, checks a document for spelling errors, the **diction** command checks language usage, and the **explain** command recommends alternate phrasing to improve your style.

Tools for Revising a Document

Since your document is stored on a disk, you can use a text editor such as **vi** to bring it into a work area, called a buffer, and change it. For example, you may add words, delete words, change words, or move text from one place to another. When you are finished, you save the document on the disk again.

Other commands, such as **cut** and **paste**, are useful if you want to move columns of text and the **awk** command is nice if you want to replace one word or phrase with another. The **awk** command is one whose name gives no clue to its function. It was named after the programmers who created it. Their last initials were a, w, and k.

In some cases, you may decide to use the **sed** stream editor to run an entire series of commands on a document.

Tools for Producing the Final Version

In this step, the instructions in the text are used to format a document. When you are ready to print a document on a printer, you use the **mm** command (or the **nroff -mm** command) and redirect the output to another file or pipe it directly to a printer. The **mm** command automatically executes the **nroff** command. For example, either of these command lines causes the "a.jones" file to be formatted and printed on the line printer:

```
$ mm a.jones | lpr
$ nroff -mm a.jones | lpr
```

When you are ready to print a document on a phototypesetter, you use the **mmt** command (or the **troff -mm** command). For example, either of these command lines causes the "a.jones" file to be formatted and printed on a phototypesetter:

```
$ mmt a.jones
$ troff -mm a.jones
```

If you have used **tbl, neqn,** or **eqn** codes in a document, you include **tbl, neqn,** and **eqn** commands in the command line. The **tbl** command formats tables, and the **eqn** command formats mathematical equations with special symbols for a phototypesetter. The **neqn** command formats mathematical equations for a printer. These commands are often called preprocessors because you format tables and equations before formatting the rest of the document. This sample command line formats a report with tables and equations and prints the report on a printer:

```
$ tbl mathreport | neqn | nroff | lpr
```

Summary

Figure 4-5 summarizes the phases of a document production project and shows some of the tools you can use.

Phase	Tools	Purpose
First draft	**ed, ex, vi**	Type a document
Checking	**diction**	Check language usage
	eqncheck	Check instructions for equations
	explain	Provide alternative phrasing
	hyphen	Find hyphenated words
	mmcheck	Check use of **mm** macros
	spell	Check spelling
	style	Analyze style
	wc	Count characters, words, lines
Revisions	**ed, ex, vi**	Edit a document
	awk	Search for patterns and replace them
	sed	Run a batch of editing commands
	cut	Cut out selected fields of each line
	paste	Merge lines of files
Final version	**eqn**	Format mathematical text for phototypesetter
	mm	Convert format instructions for printer
	neqn	Format mathematical text for printer
	nroff	Format document for printer
	troff	Format document and print on phototypesetter
	tbl	Format tables
	lpr	Print document
	mmt	Print **mm** documents on phototypesetter

Figure 4-5. Document Production Phases and Tools

Figure 4-6 gives a sample document with formatting instructions.

```
.ce
.B MEMO
.sp 2
.P
Please plan to attend a team meeting on Friday, October 19.  The agenda includes these
items:
.AL 1
.LI
Introduction of new members
.LI
Schedules
.LI
New equipment
.LI
Open items
.LE
.P
The meeting will begin at 9 A.M. and will last approximately one hour.
```

Figure 4-6. Sample Document with Formatting Instructions

The document has two **nroff/troff** codes:

- **.ce** Center the following text.
- **.sp 2** Space down two lines.

The remaining codes are **mm** macros:

- **.B** Print the following text in boldface.
- **.P** Begin a new paragraph.
- **.AL 1** Turn the following lines into a numbered list.
- **.LI** Treat as a list item.
- **.LE** End a list.

Figure 4-7 shows a document formatted according to the instructions in Figure 4-6.

MEMO

Please plan to attend a team meeting on Friday, October 19. The agenda includes these items:

1. Introduction of new members
2. Schedules
3. New equipment
4. Open items

The meeting will begin at 9 A.M. and will last approximately one hour.

Figure 4-7. Sample Formatted Document

---

intel®

CHAPTER 5
PROGRAMMING

This chapter describes how XENIX supports users writing programs. The XENIX features described are included in the XENIX 286 Extended System (except for the shell **sh**) and are not provided with the XENIX 286 Basic System. The shell **sh** is part of the Basic System. The XENIX programming environment includes

- The C programming language, a simple, flexible, efficient, and powerful tool for writing portable programs.
- Standard function libraries that provide standard ways for C programs to handle a variety of tasks, from I/O to complex computations.
- Supporting tools, a complete programming environment that includes a program checker, and a debugger, and also tools for automated translation, version control, and building new languages.
- XENIX shells that provide a structured programming language that can use all the shell's special capabilities for controlling files and processes.
- XENIX features that allow users to modify or extend XENIX to meet their special requirements.

More information on these topics is contained in the following publications:

- *The C Programming Language* by Brian W. Kernighan and Dennis M. Ritchie describes C.
- *XENIX 286 C Library Guide* describes the standard function libraries, including all kernel system calls.
- *XENIX 286 Programmer's Guide* describes the supporting tools for programmers and the **csh** shell program.
- *XENIX 286 User's Guide* and *XENIX 286 Reference Manual* describe the **sh** shell program.
- *XENIX 286 Installation and Configuration Guide* and *XENIX 286 Device Driver Guide* describe how users can modify and extend XENIX.

Appendix D gives ordering information for all these publications.

C Programming Language

This section describes the C programming language, a simple, flexible, efficient, and powerful tool for writing portable programs. C and the UNIX operating system were designed together; almost all of XENIX (and UNIX) is written in C. Before UNIX, most operating systems were written in machine-dependent assembly language. Some widely-used systems, such as CP/M-80, still are. C is a major reason for the relatively high quality of the XENIX and UNIX operating systems, and for the availability of XENIX or UNIX on so many different processors.

A C program is largely made up of a number of functions. A function takes zero or more parameters and may return a result to its caller. Parameters or results can be either values or addresses of variables in memory. For example, a function to compute square roots would take a single value parameter and return the square root as a resulting value. A function to search a string for an occurrence of a substring would take two address parameters, the addresses of the string to be searched and the string to be searched for; this function would return the address of the first occurrence of the substring in the string being searched, or return a special **NULL** value if no occurrence was found. A function can also have a variable number of parameters. For example, a function that writes formatted output can accept as parameters any number of values to be formatted.

A very powerful but simple feature of C is that it allows variables and parameters to hold function addresses and to be used to call functions. For example, a plotting function can be defined that draws a graph of some other arbitrary function, e.g., any function with a single real argument and a single real result. The address of the function to be plotted can be passed as a parameter to the plotting function.

C provides a range of data types including **char** (a single byte, often used to hold a character), signed and unsigned integers of various lengths, single-precision and double-precision floating-point numbers, and pointers to any other data type. A value that is a pointer to another type either contains the address of a value of the other type or has the special value **NULL.**

C data structures are constructed using pointers, arrays, unions, and structures. A structure is a record containing a number of fields. Each field has a distinct name and its own type. For example, a structure defining a data type "date" could include fields named "year", "month", and "day", with types **int, char,** and **char** respectively. (The **char** data type is used because only a byte of storage is needed for each of "month" and "day".) A union can contain values of different types at different times. For example, a union can be defined that will contain either an integer or a floating-point value, but not both at the same time.

An array in C contains a number of elements of the same data type. All arrays are indexed from 0 to (N-1), where N is the number of elements. An array reference in C consists of the address of the beginning of the array; because of this, C functions naturally can handle dynamic arrays (in which the number of elements is not known until run-time) as well as static arrays (in which the number of elements is known at compile-time). However, C programmers should take care to check array operations to prevent array addressing errors, as the C compiler does not generate such checking for you. Array operations in C are very simple and efficient because of the explicit use of pointers to implement arrays. For example, accessing all elements of an array in turn can be done by simply incrementing a pointer that initially references the first element of the array.

C's control structures include **if** and **switch** conditional statements (**switch** is similar to the "case" statement of some other languages), loops with tests at top or bottom of the loop, and a **for** looping statement for more complex loops, such as those with index variables. These structures provide complete support for "structured programming" methods. C also provides statements for exiting or continuing a loop from within a nested statement. The **goto** statement is also provided.

C provides many operators for forming expressions, one source of its power. Operators include arithmetic, relational, and logical operators. Also provided are bit-wise Boolean operators, left and right shift operators, and increment and decrement operators. Assignment is treated as an operator, allowing assignments to be embedded in expressions. A conditional operator evaluates one of two expressions based on the value of a third, eliminating the need for many conditional statements and often generating more efficient and more readable code.

Several capabilities are added to C by the C preprocessor, the first pass of a C compiler, which allows the user to define symbolic constants and macros and to include separate files of declarations or procedures. A macro can be used like a function but generates faster (but potentially space-consuming) "in-line" code rather than a subroutine call when it is invoked.

Despite all these features, C is simpler than many other high-level languages. A comparison to one competing language, Pascal, may be of interest. C does not provide the set structures, file structures, or variant records of Pascal, though equivalent constructs can be created in C. C also does not provide built-in functions for input/output, which are provided by Pascal. However, C does have several advantages. C supports dynamic arrays. C supports independent compilation, not originally part of Pascal. C I/O, via library functions, is more flexible than Pascal's built-in I/O functions. C supports system programming with more flexible type conversions, low- level operators, and more flexible manipulation of pointers. Finally, many aspects of C's design enable C programs to be very efficient, including increment, decrement, and assignment operators; conditional expressions; and the use of pointers for array operations. On the plus side for Pascal, its type checking is stricter, array operations can be safer (if the compiler generates subscript-checking code), and it has a richer set of data structures.

One goal of C is to support the writing of portable, machine-independent programs. However, some C features do behave differently on different machines. A style of C programming has evolved that imposes a few restrictions in order to make C programs much more portable. These restrictions are described in "C Language Portability" in the *XENIX 286 Programmer's Guide*.

C does not provide any built-in statements for input/output, dynamic storage allocation, string manipulation, concurrency, or exception handling. However, all these capabilities are provided by the XENIX libraries, described in the next section.

C Function Libraries

The machine-independence provided by the C language would do little good if different systems provided different functions for basic tasks such as input/output. In addition to the definition of the C language, there is a standard I/O library that is provided as part of almost every C language system. XENIX and UNIX provide these standard I/O functions that support opening, reading, writing, closing, and random access for files and devices; formatted I/O; and stream I/O that provides a level of buffering between the program and the operating system.

Additional standard functions have been defined over a period of several years for UNIX systems and are provided with XENIX as well. Some of these functions correspond to system calls, functions implemented by calling the XENIX kernel. The system call interface makes the transition between user code and privileged kernel code, for sensitive operations that involve processes, files, devices, or other objects managed by the kernel. The details of the system call interface are not visible to the library user, who uses a system call like any other C library function. Facilities other than I/O provided by the function libraries include

- Process control operations
- File system operations
- Interprocess communication
- Exception-handling and error-handling operations
- Character and string functions
- Dynamic memory allocation
- Computation and numeric formatting
- Screen operations, including window operations
- Encryption and decryption
- Data base record retrieval
- Searching and sorting

All these functions are described in the *XENIX 286 C Library Guide.*

Supporting Tools

A programming language, compiler, and function libraries are only some of the useful programming tools provided by XENIX. Other tools of interest are

lint a C program checker. **lint** examines C source files and warns of constructs that can cause run-time errors in C programs. Such constructs include unknown values in variables, unreachable statements, infinite loops, inconsistent types, and several others.

adb a simple machine-level debugger. You can use breakpoints or single stepping to interrupt your program and read and write memory when your program is stopped.

make automates program creation (compiling, assembling, linking) using "makefiles" that you create. A makefile lists the output files to be created, the commands that create them, and the input files from which to create them. **make** can use such a makefile to update an entire programming project with a single command. **make** checks file dates and only updates those files that must be changed.

SCCS Source Code Control System. Controls multiple versions of programs or other documents. Multiple versions can be stored in a single file, with SCCS able to recreate any version on command.

lex, yacc tools for building language translators. **lex** builds a lexical analyzer from user-supplied rules. **yacc** (yet another compiler-compiler) takes as input a set of syntactic rules along with semantic actions to be performed on recognizing the associated syntactic construct. **yacc** generates a parser to recognize the syntactic productions and perform the appropriate semantic actions. This **yacc** output is itself a language compiler. A compiler- compiler is thus a program that generates a compiler from a set of rules describing the language to be compiled.

All of these tools and the C compiler **cc** are described in the *XENIX 286 Programmer's Guide.*

Shell Programming

XENIX provides two shell programs that incorporate programming capabilities, **sh** (Bourne shell) and **csh** (C shell). These shells give you a high-level procedural language in which to communicate with XENIX, allowing you to easily perform tasks that are difficult in many operating systems. With the shell programming capabilities, commands can be

- Combined to form new commands
- Passed parameters
- Added or renamed by the user
- Arranged in series, in conditional control structures, or in looping control structures

The shells provide special support for pattern matching in file names (recognizing patterns such as "*.c"), for process control, and for I/O control. Commands can redirect input and output to and from files, terminals, other devices, or other commands. These special shell capabilities often make it easier for you to write a command as a shell procedure instead of as a C program.

sh is described in the *XENIX 286 User's Guide* and the *XENIX 286 Reference Manual.* **csh** is described in the *XENIX 286 Programmer's Guide.*

Modifying and Extending XENIX

XENIX is designed as an "open system," one that allows users to include and exclude modules and features with great flexibility. The only part of the system that cannot be easily changed by a user is the XENIX kernel, which implements a standard set of system calls that perform operating system tasks. System administrators can delete, replace, or add command programs on their systems. New command programs can be written using a shell, C, or some other programming language. Even the shell program that communicates with users can be replaced, and XENIX users can choose between different shells.

Though the kernel should not be changed directly, many aspects of the kernel are configurable, as described in the *XENIX 286 Installation and Configuration Guide.* For example, a new kernel can be created that allows for a lesser or greater number of various types of kernel objects, such as processes and locks, or that allows for a lesser or greater number of disk buffers in main memory.

Customers interfacing new hardware to XENIX systems can add device drivers, as described in the *XENIX 286 Device Driver Guide.* XENIX defines a standard and relatively simple functional interface for device drivers. As much of the work as possible is done by the kernel, with the driver supplying the device-dependent functions for initialization, opening, reading, writing, closing, and interrupt-handling for the device. The kernel also provides several utility routines that help the device driver perform common tasks, such as buffering characters or sorting disk requests to minimize access time.

intel®

APPENDIX A
BASIC SYSTEM COMMANDS

Basic System Commands by Category

The Basic System has many commands. These are organized by category in Figure A-1 and defined in the following pages.

SYSTEM ADMINISTRATION	SYSTEM STATUS	SYSTEM COMMUNICATION	FILE DISPLAY	FILE MANAGEMENT	PROGRAM EXECUTION	OFFICE TOOLS
acctcom	atq	cu	banner	cd	at	bc
accton	date	netutil	cat	chgrp	atrm	cal
asktime	finger	rcp	hd	chmod	cron	dc
chroot	ps	remote	head	chown	echo	learn
config	pstat	uucp	look	copy	env	mail
dump	uname	uulog	more	cp	expr	random
dumpdir	who	uux	nl	cpio	false	rmail
fsck	whodo		od	crypt	getopt	units
grpcheck		**FILE**	pcat	dd	kill	write
haltsys	**DEVICES**	**COMPARISON**	pr	dirname	line	
instl			tail	file	nice	**USER**
mkfs	assign	bdiff		find	nohup	**ACCESS**
mknod	deassign	cmp	**FILE**	l	rsh	
mkuser	devnm	comm	**MANIPULATION**	lc	sh	id
ncheck	df	diff		ln	sleep	login
pwadmin	disable	diff3	awk	ls	tee	logname
pwcheck	dtype	dircmp	basename	mkdir	test	newgrp
quot	du	egrep	bfs	mv	true	passwd
restor	enable	fgrep	csplit	pack	vsh	
rmuser	format	grep	ed	pwd	wait	
sddate	lpr	sdiff	ex	rm	xargs	
shutdown	mesg	uniq	join	rmdir	yes	
su	mount	what	sed	settime		
sum	setmnt		sort	touch		
sync	stty		split	umask		
sysadmin	tar		tr	unpack		
wall	tset		vi	wc		
	tty					
	umount					

F-0320

Figure A-1. Summary of Basic System Commands by Category

Alphabetical List of Commands

The commands in the Basic System are listed below in alphabetical order. Those commands that are new to this release are marked with an asterisk (*).

acctcom* search and print accounting files
accton turn system accounting on and off
asktime set system date and time
assign* assign a device to a user
at execute commands at a later time
atq* examine the "at" job queue
atrm* remove a job from the "at" job queue
awk pattern scanning and processing language
banner* print large letters
basename strip file name affixes
bc arbitrary-precision arithmetic language
bdiff* compare very large files
bfs* scan big files
cal print calendar
cat concatenate and print files
cd change working directory
chgrp change group
chmod change mode (change access permissions)
chown change file owner
chroot* change the process root directory
cmp compare two files (any type)
comm select or reject lines common to two sorted files
copy copy groups of files
cp copy
cpio* copy file archives in and out
cron execute commands at specified times
crypt encode or decode a file
csplit* split files according to context
cu call the XENIX system
date print and set the date
dc desk calculator
dd convert and copy a file
deassign* deassign a device
devnm* identify device name
df report the number of free disk blocks
diff compare two text files
diff3 compare three text files
dircmp* compare directories
dirname* deliver the directory part of a path name
disable turn terminal use off
dtype* print disk type (such as xenix, msdos, tar)
du summarize disk use
dump perform incremental file system backup
dumpdir print the names of files on a dump tape
echo echo arguments
ed invoke text editor (line editor)
egrep search a file for a pattern
enable turn terminal use on
env* set or print the environment for command execution

ex	text editor (line editor)
expr	evaluate arguments as an expression
false	provide truth value by returning with a nonzero exit code
fgrep	search a file for a pattern
file	determine file type
find	find files
finger	find information about users
format	format a disk
fsck	check file system for consistency and repair if necessary
getopt*	parse command options
grep	search a file for a pattern
grpcheck*	check group file
haltsys	shut system down
hd*	give hex dump of a file
head	give first few lines of a file
id*	print user and group ID and name
instl	install XENIX
join	join two relations
kill	terminate a process
l	list directory contents in long form (equivalent to **ls -l**)
lc	list directory contents in columns
learn	give computer-aided instruction about XENIX
line*	read one line
ln	make a link to a file
login	give access to the system
logname*	get login name
look	find files in a sorted list
lpr	send files to the line printer queue for printing
ls	list the contents of a directory
mail	send, receive, or dispose of mail
mesg	permit or deny messages sent to a terminal
mkdir	make a directory
mkfs	make a file system
mknod	make a special file
mkuser	add a new user account
more	display a file one screen at a time
mount	attach a file system to a directory on the root subtree
mv	move or rename files and directories
ncheck	generate path names from inode numbers
netutil*	administer a mail network
newgrp	log into a new group
nice	run a command at a different priority
nl*	add line numbers to a file
nohup	run background process after user logs off
od	display files in octal format
pack*	compress files
passwd	change login password
pcat*	look at packed files
pr	print a file
ps	report process status
pstat	print system facts
pwadmin*	administer aging of passwords
pwcheck*	check the password file
pwd	print the name of the working directory
quot	summarize file system ownership

random	generate a random number
rcp*	copy files between machines
remote*	execute commands on another machine
restor	invoke incremental file system restorer
rm	remove a file
rmail	send mail among users
rmdir	remove a directory
rmuser	remove a user
rsh*	invoke a restricted shell
sddate	print and set backup dates
sdiff*	compare two files side by side
sed	invoke stream editor
setmnt*	establish a mount table (**/etc/mnttab**)
settime	change file access and modification dates
sh	invoke the Bourne shell
shutdown	shut down the system
sleep	suspend execution for an interval
sort	sort or merge files
split	split a file into pieces
stty	set terminal options
su	make the user root or another user temporarily
sum	calculate checksum and count blocks in a file
sync	update the super block
sysadmin*	perform file system backup and restore
tail	deliver last part of a file
tar	archive files
tee	create a tee in a pipe to save intermediate output
test	test conditions
touch	update file access and modification times
tr	translate characters
true	return with a zero exit value
tset	set terminal type
tty	get terminal name
umask	set default file creation mask
umount	detach a file system from the root directory
uname*	print the current XENIX name
uniq	report repeated lines in a file
units	convert units
unpack*	unpack packed files
uucp	copy files from XENIX to XENIX
uulog	copy files from XENIX to XENIX
uux	execute commands on remote XENIX
vi	invoke a screen-display editor based on **ex**
vsh	invoke the visual shell
wait	wait for background jobs to finish
wall	write to all users
wc	count lines, words, and characters
what	identify files
who	list users currently logged on
whodo*	show who is doing what
write	send a message to a user's terminal
xargs*	construct argument lists and execute commands
yes	print string repeatedly

intel®

APPENDIX B
TEXT FORMATTING COMMANDS

Text Formatting Commands

This section has an alphabetical list of the commands that are part of the Text Formatting package included in the Extended System. Commands that are new to this release are marked with an asterisk (*).

col	approximate vertical motions
cut*	cut out selected fields of each line
cw*	prepare constant-width text for **troff**
cwcheck*	check **cw** macro text
deroff	remove **nroff, troff, tbl,** and **eqn** constructs
diction*	comment on writing style
diffmk*	mark differences between two versions of a file
eqn	format mathematical text for **nroff** or **troff**
eqncheck*	check mathematical text for **nroff** or **troff**
hyphen*	find hyphenated words
mm*	print documents formatted with the **mm** macros
mmcheck*	check use of **mm** macros
mmt*	typeset documents for **troff**
neqn	format mathematical text for **nroff** or **troff**
nroff	format text for a line printer or daisy wheel printer
paste*	merge lines of files
prep	prepare text for statistical processing
ptx	generate a permuted index
soelim*	expands **nroff .so** statements
spell	find spelling errors
style*	comment on writing style
tbl	format tables for **nroff** or **troff**
troff	print document on a phototypesetter

intel®

APPENDIX C
PROGRAMMING TOOLS

Programming Commands

This section has an alphabetical list of the commands that are part of the Software Development package included in the Extended System. Commands that are new to this release are marked with an asterisk (*).

adb	invoke a general-purpose debugger
admin	create and administer SCCS files
ar	maintain archives and libraries
as	invoke the XENIX assembler
cb	beautify C programs
cc	invoke the C compiler
cdc*	change the delta commentary of an SCCS delta
comb	combine SCCS deltas
cref*	make a cross-reference list
csh	invoke the C shell (a command interpreter with C-like syntax)
ctags	create a tags file
delta	make a delta (change) to an SCCS file
doscat*	concatenate a file on an MS-DOS flexible disk
doscp*	copy files to or from MS-DOS flexible disks
dosdir*	list the directory of an MS-DOS flexible disk
dosls*	list the directory of an MS-DOS flexible disk
dosmkdir*	create an MS-DOS directory on an MS-DOS flexible disk
dosrm*	delete an MS-DOS file
dosrmdir*	delete an MS-DOS directory
get	get a version of an SCCS file
gets	get a string from the standard input
hdr*	display selected parts of object files
help	ask for help about SCCS commands
ld	invoke the link editor
lex	generate programs for lexical analysis
lint	check C language usage and syntax
lorder	find ordering relation for an object library
m4	invoke a macro processor
make	maintain, update, and regenerate groups of programs
mkstr	create an error message file from C source
nm	print a name list
prof	display profile data
prs	print an SCCS file
ranlib	convert archives to random libraries
ratfor	convert rational FORTRAN into standard FORTRAN
regcmp*	compile regular expressions
rmdel*	remove a delta from an SCCS file
sact*	print current SCCS file editing activity

sccsdiff	compare two versions of an SCCS file
size	print the size of an object file
spline	interpolate a smooth curve
stackuse*	determine stack requirements for C programs
strings	find the printable strings in a binary file
strip	remove symbols and relocation bits from an object file
time	time a command
tsort	sort a file topologically
unget	undo a previous get of an SCCS file
val	validate an SCCS file
xref*	cross-reference C programs
xstr	extract strings from C programs
yacc	invoke a compiler-compiler (yet another compiler-compiler)

Standard C Libraries

The following libraries are provided with the Extended System. In some cases, versions for small, middle, and large model programs are included, and in other cases only the version for the small model is provided. These are the standard libraries:

libc This is the standard library that contains all system call interfaces, standard I/O routines, and other general purpose services. Versions for small, middle, and large models are provided.

libm This is the standard math library. Versions for small, middle, and large models are provided.

libl This library is for use with programs produced by **lex.** A version for the small model is provided.

liby This library is for use with programs produced by **yacc.** A version for the small model is provided.

libtermcap This library has routines for accessing the **termcap** data base of terminal characteristics. Versions for small, middle, and large models are provided.

libtermlib This library is the same as **libtermcap.** Both **libtermcap** and **libtermlib** link to the same file. Both names are kept for historical reasons. Versions for small, middle, and large models are provided.

libcurses This library has routines for manipulating the screen and cursor. Versions for small, middle, and large models are provided.

libdbm This library has data base management routines. Versions for small, middle, and large models are provided.

The functions provided with the standard C libraries are listed below. Those that are new to this release are marked with an asterisk (*).

The Standard C Library -- libc

This library also includes all system functions, listed separately at the end of this appendix.

_tolower	convert to lowercase
_toupper	convert to uppercase
a64l*	convert base-64 ASCII to long integer
abort	generate an IOT fault
abs	integer absolute value
asctime	convert time data to ASCII
assert	program verification
atof	convert ASCII string to floating number
atoi	convert ASCII string to integer
atol	convert ASCII string to long integer
bsearch*	binary search
calloc	allocate memory
clearerr	clear error
crypt	DES (Data Encryption Standard) encryption
ctermid*	generate file name for terminal
ctime	convert time to ASCII string
cuserid*	character login name of user
defopen	open default parameter file
defread	read default parameters
ecvt	format conversion
encrypt	DES (Data Encryption Standard) encryption
endgrent	close group file
endpwent	close password file
fclose	close a stream
fcvt	format conversion
fdopen	reopen a stream
feof	test for end of file
ferror	test for error
fflush	flush a stream
fgetc	get character from a stream
fgets	get a string from a stream
fileno	convert a stream number to a file descriptor
fopen	open a stream
fprintf	formatted output routine
fputc	write a character to a stream
fputs	write a string to a stream
fread	buffered input
free	free memory
freopen	reopen a stream
frexp	return mantissa
fscanf	formatted input conversion
fseek	seek within a stream
ftell	obtain file pointer position
fwrite	buttered output
fxlist	get name list entries from a file
gcvt	format conversion
getc	get a character from a stream

getchar	get a character from a stream
getcwd*	get path name of current working directory
getenv	get a value for an environment variable
getgrent	get group file entry
getgrgid	get group file entry
getgrnam	get group file entry
getlogin	get login name
getopt*	parse command line options
getpass	read a password
getpw	get a name from the user ID
getpwent	get a password file entry
getpwnam	get a password file entry
getpwuid	get a password file entry
gets	get a string from a stream
getw	get a word from a stream
gmtime	obtain Greenwich Mean Time information
gsignal*	raise a software signal
isalnum	test for alphanumeric
isalpha	test for alphabetic character
isascii	test for ASCII character
isatty	check for terminal
iscntrl	test for control character
isdigit	test for digit
isgraph	test for printing character
islower	test for lowercase
isprint	test for printing character
ispunct	test for punctuation
isspace	test for space
isupper	test for uppercase
isxdigit	test for hex digit
l3tol	convert 3-byte integer to long
l64a*	convert a long integer to base-64 ASCII
ldexp	load exponent of floating point number
localtime	obtain local time information
logname	get login name of a user
longjmp	nonlocal goto
lsearch*	linear search and update
ltol3	convert long to 3-byte integer
malloc	allocate memory
mktemp	make a temporary file
modf	return a fractional part
monitor	prepare an execution profile
nlist	get entries from the name list
pclose	close pipe to process
perror	print system error messages
popen	initiate I/O to or from a process
printf	formatted output routine
putc	write a character to a stream
putchar	write a character to a stream
putpwent*	write a password file entry
puts	write a string to a stream
putw	write a word to a stream
qsort	quick sort routine
rand	random number generator

realloc	reallocate memory
regcmp	regular expression compile
regex	regular expression execute
rewind	seek to the beginning of a file
scanf	formatted input conversion
setbuf	assign buffering to a stream
setfrent	rewind a group file pointer
setjmp	nonlocal goto
setkey	DES (Data Encryption Standard) encryption
setpwent	rewind password file pointer
sleep	suspend execution for an interval
sprintf	formatted output routine
srand	seed random number generator
sscanf	formatted input conversion
ssignal*	software signal
strcat	concatenate strings
strchr*	find a character in a string
strcmp	compare strings
strcpy	copy strings
strcspn*	find the length of a substring
strlen	get string length
strncat	concatenate strings
strncmp	compare strings
strncpy	copy strings
strpbrk	find a string in a string
strrchr*	find a character in a string
strspn*	find the length of a substring
strtok*	find a token within a string
swab	swap bytes
system	execute a shell command
tmpfile*	create a temporary file
tmpnam*	create a temporary file name
toascii	convert to ASCII
tolower	convert to lowercase
toupper	convert to uppercase
ttyname	find the name of a terminal
tzset*	set external time variables
ungetc	push a character back onto a stream
xlist	get name list entries from a file

The Standard Math Library -- libm

acos	arc cosine function
asin	arc sine function
atan	arc tangent function
atan2	arc tangent function
cabs	Euclidean distance
ceil	ceiling value
cos	cosine function
cosh	hyperbolic cosine
exp	exponentiation
fabs	returns |x|
floor	whole number at or immediately below its argument
fmod*	remainder function

gamma*	log gamma function
hypot	sqrt(x*x + y*y)
j0	Bessel function
j1	Bessel function
jn	Bessel function
log	natural logarithm
log10	log base 10
pow	power function
sin	sine function
sinh	hyperbolic sine
sqrt	square root function
tan	tangent function
tanh	hyperbolic tangent
y0	Bessel function
y1	Bessel function
yn	Bessel function

The Default lex Library -- libl (small model only)

main	**lex** program entry
yyless	**lex** routine to "unget" source characters
yywrap	**lex** end of file routine

The Default yacc Library -- liby (small model only)

main	**yacc** program entry
yyerror	**yacc** error handler

The Terminal Capabilities Library -- libtermcap (libtermlib)

tgetent	get terminal capability entry
tgetflag	test for presence of capability
tgetnum	get numeric value of capability
tgetstr	get string value of capability
tgoto	get cursor addressing string
tputs	decode padding information

The Screen Manipulation Library -- libcurses

many screen and cursor manipulation routines

The Data Base Management Library -- libdbm

dbminit	open data base
delete	delete key in data base
fetch	access key in data base
firstkey	get first key in data base
nextkey	get next key in data base
store	store key in data base

System Calls

The Software Development package includes the following system calls. Those that are new to this release are marked with an asterisk (*).

access	determine accessibility of a file
acct	enable or disable process accounting
alarm	set a process's alarm clock
chdir	change working directory
chmod	change mode of a file
chown	change the owner and group of a file
chroot	change the root directory
chsize*	change the size of a file
close	close a file descriptor
creat	create a new file or rewrite an existing one
creatsem	create an instance of a binary semaphore
dqoverlay*	load overlay for UDI-based (Universal Development Interface) programs
dup	duplicate an open file descriptor
dup2	duplicate an open file descriptor
execl	execute a file
exit	terminate a process
fcntl*	file control
fork	create a new process
fstat	get file status
ftime	get system time
getegid	get effective group ID
geteuid	get effective user ID
getgid	get group ID
getpgrp*	get process group
getpid	get process ID
getppid	get parent process ID
getuid	get real user ID
ioctl	control device
kill	send a signal to a process or a group of processes
link	link to a file
lock*	lock a process in memory
locking	lock or unlock a file region for reading or writing
lseek	move a read/write file pointer
mknod	make a file
mount	mount a file structure
nap*	sleep for a short time
nice	change the priority of a process
open	open a file for reading or writing
opensem	open a semaphore
pause	suspend process until signal
pipe	create an interprocess channel
profil	execution time profile
ptrace	process trace
rdchk	check if there is data to be read
read	read from a file
sbrk	change data segment space allocation

sdenter*	enter a shared data region
sdfree*	release a shared data region
sdget*	attach to a shared data region
sdgetv*	synchronize the use of shared data
sdleave*	leave a shared data region
sdwaitv*	synchronize use of shared data
setgid	set group ID
setpgrp*	set process group ID
setuid	set user ID
shutdn	flush block I/O and halt system
signal	specify what to do on receipt of a signal
sigsem	signal a process waiting on a semaphore
stat	get file status
stime	set time
sync	update the super block
time	get time
times	get process and child process times
ulimit*	get and set user limits
umask	get and set file creation mask
umount	unmount a file system
uname*	get name of current XENIX system
unlink	remove a directory entry
ustat*	get file system statistics
utime	set file access and modification times
wait	wait for a child process to stop or terminate
waitsem	wait for a semaphore
write	write on a file

intel®

APPENDIX D
RELATED PUBLICATIONS

Related Intel Publications

Copies of the following publications can be ordered from

Literature Department
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

Overview of the XENIX 286 Operating System, Order Number 174385 -- XENIX history, XENIX uses, basic XENIX concepts, and an overview of other XENIX manuals.

XENIX 286 Installation and Configuration Guide, Order Number 174386 -- how to install XENIX on your hardware and tailor the XENIX configuration to your needs.

XENIX 286 User's Guide, Order Number 174387 -- a tutorial on the most-used parts of XENIX, including terminal conventions, the file system, the screen editor, and the shell.

XENIX 286 Visual Shell User's Guide, Order Number 174388 -- a XENIX command interface ("shell") that replaces the standard command syntax with a menu-driven command interpreter.

XENIX 286 System Administrator's Guide, Order Number 174389 -- how to perform system administrator chores such as adding and removing users, backing up file systems, and troubleshooting system problems.

XENIX 286 Communications Guide, Order Number 174461 -- installing, using, and administering XENIX networking software.

XENIX 286 Reference Manual, Order Number 174390 -- all commands in the XENIX 286 Basic System.

XENIX 286 Programmer's Guide, Order Number 174391 -- XENIX 286 Extended System commands used for developing and maintaining programs.

XENIX 286 C Library Guide, Order Number 174542 -- standard subroutines used in programming with XENIX 286, including all system calls.

XENIX 286 Device Driver Guide, Order Number 174393 -- how to write device drivers for XENIX 286 and add them to your system.

XENIX 286 Text Formatting Guide, Order Number 174541 -- XENIX 286 Extended System commands used for text formatting.

Suggested Readings

The popularity of XENIX and other UNIX-like operating systems has caused many new books to appear in the bookstores. You may want to supplement the XENIX documentation with one or more of these books:

- Banahan, Mike, and Andy Rutter. *The UNIX Book*. New York: John Wiley & Sons, Inc., 1983.
- Bourne, S. R. *The UNIX System*. Reading, Mass.: Addison-Wesley Publishing Company, 1982.
- Christian, Kaare. *The UNIX Operating System*. New York: John Wiley & Sons, Inc. 1983.
- Groff, James R., and Paul N. Weinberg. *Understanding UNIX: A Conceptual Guide*. Indianapolis, Indiana: Que Corporation, 1983.
- Kernighan, Brian W., and Rob Pike. *The UNIX Programming Environment*. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1984.
- Kernighan, Brian W., and Dennis M. Ritchie. *The C Programming Language*. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1978.
- McGilton, Henry, and Rachel Morgan. *Introducing the UNIX System*. New York: McGraw-Hill Book Company, 1983.
- Sobell, Mark G. *A Practical Guide to the UNIX System*. Menlo Park, California: The Benjamin/Cummings Publishing Company, Inc., 1984.
- Thomas, Rebecca, and Jean Yates. *A User Guide to the UNIX System*. Berkeley, Calif.: OSBORNE/McGraw-Hill, 1982.
- Yates, Jean, and Sandra L. Emerson. *The Business Guide to the UNIX System*. Reading, Mass.: Addison-Wesley Publishing Company, 1984.

intel®

INDEX

intel®

Overview of the
XENIX 286 Operating System
174385-001

REQUEST FOR READER'S COMMENTS

Intel's Technical Publications Departments attempt to provide publications that meet the needs of all Intel product users. This form lets you participate directly in the publication process. Your comments will help us correct and improve our publications. Please take a few minutes to respond.

Please restrict your comments to the usability, accuracy, organization, and completeness of this publication. If you have any comments on the product that this publication describes, please contact your Intel representative. If you wish to order publications, contact the Literature Department (see page ii of this manual).

1. Please describe any errors you found in this publication (include page number).

2. Does this publication cover the information you expected or required? Please make suggestions for improvement.

3. Is this the right type of publication for your needs? Is it at the right level? What other types of publications are needed?

4. Did you have any difficulty understanding descriptions or wording? Where?

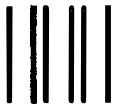
5. Please rate this publication on a scale of 1 to 5 (5 being the best rating). _____

NAME _____ DATE _____
TITLE _____
COMPANY NAME/DEPARTMENT _____
ADDRESS _____
CITY _____ STATE _____ ZIP CODE _____
(COUNTRY)

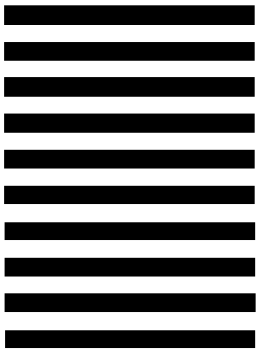
Please check here if you require a written reply. ☐

WE'D LIKE YOUR COMMENTS ...

This document is one of a series describing Intel products. Your comments on the back of this form will help us produce better manuals. Each reply will be carefully reviewed by the responsible person. All comments and suggestions become the property of Intel Corporation.



NO POSTAGE
NECESSARY
IF MAILED
IN U.S.A.



BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 79 BEAVERTON, OR

POSTAGE WILL BE PAID BY ADDRESSEE

Intel Corporation
5200 N.E. Elam Young Parkway.
Hillsboro, Oregon 97123

ISO-N TECHNICAL PUBLICATIONS

INTEL CORPORATION, 3065 Bowers Avenue, Santa Clara, California 95051 (408) **987-8080**

Printed in U.S.A.

SOFTWARE

0026/44K/0984/OSPS