AT&T

UNIX® SYSTEM V RELEASE 4

Migration Guide

UNIX Software Operation

UNIX® SYSTEM V RELEASE 4

Migration Guide

UNIX Software Operation

Published by Prentice-Hall, Inc.
A Division of Simon & Schuster
Englewood Cliffs, New Jersey 07632

IMPORTANT NOTE TO USERS

While every effort has been made to ensure the accuracy of all information in this document, AT&T assumes no liability to any party for any loss or damage caused by errors or omissions or by statements of any kind in this document, its updates, supplements, or special editions, whether such errors are omissions or statements resulting from negligence, accident, or any other cause. AT&T further assumes no liability arising out of the application or use of any product or system described herein; nor any liability for incidental or consequential damages arising from the use of this document. AT&T disclaims all warranties regarding the information contained herein, whether expressed, implied or statutory, *including implied warranties of merchantability or fitness for a particular purpose.* AT&T makes no representation that the interconnection of products in the manner described herein will not infringe on existing or future patent rights, nor do the descriptions contained herein imply the granting or license to make, use or sell equipment constructed in accordance with this description.

AT&T reserves the right to make changes without further notice to any products herein to improve reliability, function, or design.

TRADEMARKS

NeWS and SunOS are registered trademarks of Sun Microsystems, Inc.
OPEN LOOK is a trademark of AT&T.
PostScript is a registered trademark of Adobe Systems, Inc.
UNIX is a registered trademark of AT&T.
VAX is a trademark of Digital Equipment Corporation.
XENIX is a registered trademark of Microsoft Corporation.

10 9 8 7 6 5 4 3 2 1

ISBN 0-13-933821-7

UNIX
PRESS
A Prentice Hall Title

P R E N T I C E H A L L

ORDERING INFORMATION

UNIX® SYSTEM V, RELEASE 4 DOCUMENTATION

To order single copies of UNIX® SYSTEM V, Release 4 documentation, please call (201) 767-5937.

ATTENTION DOCUMENTATION MANAGERS AND TRAINING DIRECTORS:
For bulk purchases in excess of 30 copies please write to:
Corporate Sales
Prentice Hall
Englewood Cliffs, N.J. 07632.
Or call: (201) 592-2498.

ATTENTION GOVERNMENT CUSTOMERS: For GSA and other pricing information please call (201) 767-5994.

Prentice-Hall International (UK) Limited, *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Prentice-Hall Canada Inc., *Toronto*
Prentice-Hall Hispanoamericana, S.A., *Mexico*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Simon & Schuster Asia Pte. Ltd., *Singapore*
Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*

AT&T UNIX® System V Release 4

General Use and System Administration

UNIX® System V Release 4 Network User's and Administrator's Guide
UNIX® System V Release 4 Product Overview and Master Index
UNIX® System V Release 4 System Administrator's Guide
UNIX® System V Release 4 System Administrator's Reference Manual
UNIX® System V Release 4 User's Guide
UNIX® System V Release 4 User's Reference Manual

General Programmer's Series

UNIX® System V Release 4 Programmer's Guide: ANSI C and Programming Support Tools
UNIX® System V Release 4 Programmer's Guide: Character User Interface (FMLI and ETI)
UNIX® System V Release 4 Programmer's Guide: Networking Interfaces
UNIX® System V Release 4 Programmer's Guide: POSIX Conformance
UNIX® System V Release 4 Programmer's Guide: System Services and Application Packaging Tools
UNIX® System V Release 4 Programmer's Reference Manual

System Programmer's Series

UNIX® System V Release 4 ANSI C Transition Guide
UNIX® System V Release 4 BSD / XENIX® Compatibility Guide
UNIX® System V Release 4 Device Driver Interface / Driver–Kernel Interface (DDI / DKI) Reference Manual
UNIX® System V Release 4 Migration Guide
UNIX® System V Release 4 Programmer's Guide: STREAMS

Contents

1. ABOUT THIS GUIDE

1. ABOUT THIS GUIDE

Introduction

Many of the features introduced in earlier UNIX® System V releases exist unchanged in Release 4.0. Other features have been enhanced, sometimes gradually over a number of releases, to the point where a user may need to become reacquainted with a feature and to explore its capabilities as they exist in Release 4.0. Some features introduced in earlier releases have been replaced altogether by new technology.

This guide describes UNIX System V Release 4.0 to the user who is upgrading from an earlier release of UNIX System V. It describes changes that have been made to the system as it has evolved to meet the growing needs of the UNIX system marketplace.

Chapter 1, "About this Guide" describes the audience and scope of the *Migration Guide.*

Chapter 2, "The Evolution of UNIX System V," briefly describes the history of the UNIX system so that the goals of Release 4.0 can be understood in context.

Chapter 3, "Migrating from Release 2," describes UNIX System V Release 2 and explains how the features introduced in Releases 2.0 and 2.1 have evolved through Release 3 to their present state in Release 4.0. Users upgrading from a Release 3 system can ignore Chapter 3.

Chapter 4, "Migrating from Release 3," describes features introduced in Release 3.0, 3.1, and 3.2. Users upgrading from a Release 2 system should read Chapter 4, as many of the features introduced in Release 3 are important features in Release 4.0.

Chapter 5, "UNIX System V Release 4.0," describes Release 4.0. The scope of Release 4.0 exceeds the scope of earlier releases and constitutes a redesign of some aspects of the system. Therefore, the description of Release 4.0 in Chapter 5 is not presented as a list of new features. Instead, it examines functional areas of the system, describes these functional areas as they existed in earlier UNIX System V releases, then describes the changes made in each functional area in Release 4.0. Discussion of new features is integrated into the discussion of functional areas. For example, a section entitled "System Access" describes the method for accessing a UNIX system running an earlier release of UNIX System V, then describes the architectural changes made in Release 4.0 to provide a consistent access mechanism for both local and network users. For a listing of Release 4.0 features, followed by a description of each new feature, see the "Product Overview" at the beginning of the *Product Overview and Master Index.*

2. THE EVOLUTION OF UNIX SYSTEM V

2. THE EVOLUTION OF UNIX SYSTEM V

Introduction

The UNIX system is an evolving system. Developed at Bell Laboratories in the 1960s, the UNIX system was strictly an internal system until AT&T began licensing it in the early 1970s. As a non-proprietary system, the UNIX system was distributed freely—primarily to universities, but also to equipment vendors who enhanced the system to meet the needs of their hardware. Many versions of the system began to appear, all with the same basic design, but with differences resulting from the different implementations.

In the mid-1970s, the first commercial implementation of the UNIX system appeared. Soon afterward, Microsoft Corporation introduced the XENIX® System.

In the academic and military worlds, an implementation of the system developed at the University of California at Berkeley became standard on the DEC VAX line of computers. From Berkeley's implementations (4.1, 4.2, and 4.3 BSD), other versions evolved, including Sun Microsystems' SunOS and many of the microprocessor-based versions available today.

Following a court-ordered divestiture in 1983, AT&T was allowed to enter the computer industry and market the operating system originally developed in its own laboratories. AT&T's first commercial implementation of the UNIX system was called UNIX System III. By 1984, UNIX System III had evolved into UNIX System V.

To date, there have been three major releases of UNIX System V: Release 2, Release 3, and, now, Release 4. Each release provides features that extend the power, the usefulness, and the usability of the UNIX system.

3. MIGRATING FROM RELEASE 2

3 Migrating from Release 2

Introduction

UNIX System V Release 2 comprised two major releases, which introduced new features, and several maintenance releases. The major releases were Release 2.0 and Release 2.1.

The features introduced in the major Release 2 releases are described in this section. Users upgrading from Release 2 should read this section to learn what has happened to each feature as the system has evolved through the major Release 2 and Release 3 releases; the descriptions explain the extent to which a Release 2 feature has been changed or enhanced in Release 4.0, or whether or not it has been replaced or deleted altogether.

Release 2.0 Features

Release 2.0 introduced the following features to UNIX System V:

- self-configuration
- dynamic disk partitioning
- file system hardening
- advisory file and record locking
- shell enhancements
- system administration menus

Of the features introduced in Release 2, self-configuration and dynamic disk partitioning are unchanged in Release 4.0.

Self-Configuration

Release 2.0 introduced self-configuration, a feature that enables peripheral drivers to be kept on the hard disk instead of permanently in the UNIX System V kernel. When the system is booted, the feature can automatically detect new hardware and retrieve the corresponding drivers.

Dynamic Disk Partitioning

Dynamic disk partitioning allows disk partitioning information to be kept on the disk (hard or floppy) instead of in the kernel. This feature gives the user more flexibility in partitioning the disk to suit current needs.

File System Hardening

File system hardening gives the UNIX system's file system additional protection if there is a power outage, system crash, or removal of a floppy diskette during update. File system hardening is done through ordered writes to the disk, frequent disk buffer flushings, detection of corrupt file systems when mounting, and automatic sanity flag checking on all file systems to ensure integrity.

Advisory File and Record Locking

File and record locking allows a process to lock a file (or one or more contiguous bytes of a file) for exclusive use by that process. If a file or record is locked, then access to it by another process is restricted according to the type of lock on it (either read or write).

Advisory file and record locking was introduced in Release 2.0 and supplemented in Release 2.1 with mandatory locking. Whereas advisory file and record locking relies on cooperating processes to enforce the record locking protocol, mandatory file and record locking is enforced by the system calls used to access files. Both record locking modes continue to be supported in Release 4.0. In addition, Release 4.0 supports XENIX-style file and record locking, via the `locking` function, for compatibility with existing XENIX source code.

Shell Enhancements (Job Control)

The primary enhancement to the shell in Release 2.0 was the shell layer manager, which is an implementation of job control that works with standard terminals without special hardware. Release 4.0 offers an optional shell (`jsh`) that features a new implementation of job control that conforms to the POSIX standard. The job control shell allows a user to stop and later resume jobs executing in either the foreground or the background, and to move jobs back and forth between the background and the foreground.

The shell layer manager continues to be supported in Release 4.0.

System Administration Menus (sysadm)

The System Administration Menu command (`sysadm`) and associated subcommands are menu-driven commands that simplify the job of system administration. The system administration menus introduced in 2.0 are available in Release 4.0, but the menu interface has been significantly enhanced to improve usability and expand the coverage of the interface.

Release 2.1 Features

Release 2.1 introduced two important features to UNIX System V:

- Demand paging
- Mandatory file and record locking

In addition to introducing new features, Release 2 introduced general performance improvements. For example, the operating system and many commands were recompiled in Release 2.1 with a new issue of the C Programming Language Utilities (CPLU) Issue 3 to increase the speed of floating point operations.

For the first time in Release 2.1, CPLU was removed from the UNIX System V base and made available as a separate product.

Demand Paging

REGIONS demand paging, introduced in Release 2.1, is a virtual memory management architecture that replaced the original UNIX system swapping architecture. Demand paging allows the UNIX system to execute processes that exceed the address space of main memory.

In Release 4.0, a third-generation memory management architecture replaces both the original swapping architecture and the REGIONS virtual-memory architecture. The new Virtual Memory (VM) architecture memory-management provides the benefits of demand paging, plus a number of new, distinctive advantages, such as greater portability of kernel code. System programs that reference REGIONS data structures will not work in Release 4.0. For more information about the VM architecture, see Chapter 5 of this guide.

Mandatory File and Record Locking

Mandatory File and Record Locking was introduced in Release 2.1 to supplement Advisory File and Record Locking, a Release 2.0 feature. Both modes of record locking are supported in Release 4.0, as well as XENIX-style file and record locking (provided for compatibility with XENIX source code).

For an explanation of the differences in the two modes of record locking, see the description of File and Record Locking in "Release 2.0 Features."

4. MIGRATING FROM RELEASE 3

4. MIGRATING FROM RELEASE 3

4 Migrating from Release 3

Introduction

This section is directed to Release 2 and Release 3 users who are upgrading to Release 4.0. The section describes the features that were introduced to System V in the Release 3 releases and alerts users to Release 3 features that have been replaced or significantly enhanced in Release 4.0.

UNIX System V Release 3 consisted of three releases—Release 3.0, Release 3.1, and Release 3.2. Although each release introduced a range of new features, Release 3.0 enhancements were mainly in the area of networking, Release 3.1 focused on internationalizing the UNIX system, and Release 3.2 added security enhancements.

Release 3.0 Features

Release 3.0 introduced features that are important to extending the UNIX system's capabilities as a networked operating system. These features, packaged together in the Network Support Utilities package, consisted of

- STREAMS
- the Transport Level Interface (TLI)
- the Listener.

These features continue to be an important part of the UNIX system.

Release 3.0 also introduced Remote File Sharing (RFS), an add-on package that allows users to share files and directories across a network.

Performance improvements introduced in Release 3.0 included

- Shared Libraries and Shared Library Generation
- Signal Mechanism Enhancements
- Improved Terminal Support.

Remote File Sharing

Remote File Sharing is a file-sharing package that allows users to share files, directories, devices, and named pipes transparently among computers that are linked by a network. The administrator of each computer on the network controls which local resources are available to other computers and which remote resources local users can access. Sharing is done at the directory level. When a user shares a directory, its entire contents are shared.

In Release 3.0 Remote File Sharing was tied to the Universal Receiver Protocol (URP) on 3B2 Computers that were connected via the AT&T STARLAN NETWORK; however, in later Release 3 releases, Remote File Sharing is media- and protocol-independent.

In Release 3.1, a loop-back feature was provided that enables an administrator to simulate Remote File Sharing processing within one computer. Application programs designed to use RFS can be tested partially without actually communicating with a remote computer. This feature can also be used to demonstrate RFS when only one computer is available.

Release 3.1 also introduced RFS client caching. This feature enables a client system (one that is accessing data from another system) in a Remote File Sharing arrangement to maintain a local copy of the data it needs. When a block of data is read from or written to a remote system, it is placed in a local buffer where it can be accessed by subsequent requests for data by local processes. This can reduce significantly the amount of data that needs to be sent across the network, resulting in significant performance improvements for many patterns of remote file use.

In Release 4.0, Remote File Sharing has been implemented as a file system type (`rfs`) and is one of two distributed file system types supported by a new Virtual File System architecture. The second distributed file system type is an implementation of Network File System, a SunOS file-sharing package that allows a computer to share one or more of its file systems with other computers in a heterogeneous operating system environment. Both RFS and NFS can run simultaneously on the same machine, and both packages are administered through a common command interface.

For more information about RFS, see the *Network User's and Administrator's Guide*. For more information about the Virtual File System and the Network File System, see Chapter 5 of this guide.

STREAMS

STREAMS was introduced in Release 3.0 as a mechanism and a set of tools for the development of communication and networking services within the UNIX system kernel.

STREAMS defines standard interfaces for character input/output within the kernel, and between the kernel and the rest of the UNIX system. The STREAMS mechanism enables modular, portable program development and seamless integration of network services.

The interfaces defined in STREAMS allow networking architectures and higher-level protocols to be independent of underlying protocols, drivers, and media. Higher-level services are created by selecting and connecting lower-level services and protocols.

In Release 3.0, the STREAMS mechanism was provided as an interface for implementing character I/O devices and networking protocols in the kernel. In Release 4.0, the UNIX System V terminal subsystem and pipes have been implemented to take advantage of the STREAMS mechanism.

For information about STREAMS-based ttys and pipes, see the *Programmer's Guide: STREAMS*.

Transport Level Interface

The Transport Level Interface (TLI) defines an interface between programs and protocols at the transport layer of the Open Systems Interconnection (OSI) Reference Model. TLI relieves user programs of the need to know special characteristics of underlying networking protocols. The TLI definition is implemented through the Networking Services Library (`libnsl`). User programs that are written using TLI work properly with any network transport provider that also conforms to TLI.

TLI supports two modes of transfer—connection-oriented, which transports data over an established connection in a reliable, sequenced manner, and connectionless, which supports data transfer in self-contained units (datagrams) with no logical relationship required among units.

TLI remains unchanged in Release 4.0; however, a number of network protocols are supported for the first time in Release 4.0, and all have been implemented to understand TLI.

For more information about the Networking Services Library (`libnsl`) and TLI, see the *Programmer's Guide: Networking Interfaces*.

Listener

For each transport provider on a system, UNIX System V Release 3.0 provides an active user-level program called a Listener. The purpose of the Listener is to receive requests for network services from another system, interpret which network service is needed, and initiate a process that has been designated to provide the requested network service. The Listener then drops out of the communications path and continues to listen for new service requests.

Although the Listener remains virtually unchanged in Release 4.0, it has been implemented under the Service Access Facility (SAF)—a new facility that manages all external access to the system and provides consistent handling of connection requests from different access points.

For more information about the SAF, see Chapter 5 of this guide and the *System Administrator's Guide.*

Shared Libraries

Shared libraries were introduced in Release 3.0.

A shared library is a set of routines that is attached to a program at run-time, rather than having routines combined with an application program when it is compiled. The end user of an application that was built in this way benefits in several ways:

- The application program may occupy less storage space.
- When it is running, the application program occupies less space in memory.
- When routines in a shared library are changed, the new, improved versions are accessible without recompilation of the programs that access them. For example, by improving the performance of one routine, the performance of every application that uses that routine will be improved immediately.

For information about shared libraries, see the *Programmer's Guide: System Services and Application Packaging Tools.*

getopts

A new shell function, `getopts`, was offered in Release 3.0 as a replacement for the `getopt` command. To ease migration, both commands were supported in

Release 3.0, and, to assist in the conversion of affected shell scripts, both a conversion command (`getoptcvt`) and hand conversion procedures were provided with Release 3.0.

Beginning with Release 3.1, only `getopts` is supported.

Signal Mechanism Enhancements

A new signal interface (`sigset`) was provided in Release 3.0 as an improved mechanism to manage signals. New system calls allowed a programmer to establish critical sections of code that would not be interrupted by a set of signals. These signal-handling system calls were compatible in name and calling sequences with 4.1 BSD.

In Release 4.0, UNIX System V supports the signal interface defined by POSIX P1003.1, as well as the pre-3.0 interface (`signal`) and the Release 3.0 interface (`sigset`).

Improved Facilities for Supporting Terminals

To extend terminal support in UNIX System V, Release 3.0 introduced the Windowing Utilities package, as well as enhancements to the Terminal Information Utilities package.

The Windowing Utilities package consists of software required by AT&T windowing terminals. The Terminal Information Utilities package contains a database (`terminfo`) that allows programmers to write programs to manipulate the screen displays of various terminals.

Updated versions of the Terminal Information Utilities and the Windowing Utilities are provided with Release 4.0. For a description of these and other utilities packages provided as part of Release 4.0, see the "Product Overview" in the *Product Overview and Master Index.*

Release 3.1 Features

The focus of Release 3.1 was on making UNIX System V more adaptable to different languages and national conventions. To promote the use of the UNIX system internationally, Release 3.1 offered support for 8-bit characters, alternate date and time formats, and alternate character classifications and conversion rules. (For a full discussion of the features in Release 4.0 that support international applications, see "Internationalization" in Chapter 5 of this guide.)

In addition, Release 3.1 provided a new version of `awk` (called `nawk`) that supports 8-bit characters.

Performance improvements in Release 3.1 consisted of improved recovery of files from `cpio` archives, incremental backup of nested files, the ability to page the user area, and a smaller and faster `curses` library.

Release 3.1 also introduced enhancements to RFS. For a description of the enhancements, see the description of RFS in "Release 3.0 Features" earlier in this guide.

Support for Eight-Bit Code Sets

Because the ASCII character code only uses seven of the available eight bits in a byte, some commands in the pre-3.1 command set made special use of this eighth bit; other commands assumed that if the bit was set, the byte was invalid. In Release 3.1, the `cat`, `ed`, `egrep`, `expr`, `find`, `grep`, `ls`, `pg`, `sed`, `sort`, and `vi` commands and the `curses` library were changed so that they no longer use the eighth bit of each byte. This change enables these commands to handle code sets where all eight bits are used in character encoding (to handle such things as accented vowels).

Support for eight-bit code sets in Release 3.1 paved the way for support for multiple code sets and multi-byte character representation in Release 4.0.

Support for Alternate Date and Time Formats

The `cpio`, `date`, `ls`, `mount`, `pr`, and `sort` commands were changed in Release 3.1 to provide the date and time in the language and national conventions given by the value of the `LANGUAGE` environment variable. While the United States conventions remained the default, other languages and national conventions could be supported by creating and installing a file for the language desired in the `/usr/lib/cftime` directory.

In Release 4.0, foreign languages and national conventions are supported by a new set of environment variables, which take precedence over old ones. The `/usr/lib/cftime` directory has been changed to `/usr/lib/locale`, and the structure of the directory has changed.

Support for Alternate Character Classification and Conversion Rules

`cat`, `ed`, `egrep`, `grep`, `ls`, `pg`, `sed`, `sort`, and `vi` (commands that convert characters from upper- to lowercase or classify characters as alphabetic, printable, upper- or lowercase, and so on) were changed in Release 3.1 to support code sets or classification rules according to the value of the `CHRCLASS` environment variable. While ASCII remains the default for these operations, other conversion and classification rules are supported by creating and installing a file describing these rules.

In Release 4.0, the environment variable `LC_CTYPE` takes precedence over `CHRCLASS`.

New awk (nawk)

`awk` is a programming language for information retrieval and data manipulation that can be used by people with very little programming background. Because `awk` does not provide support for 8-bit characters, a new version, called `nawk`, was introduced in Release 3.1 as part of the effort to internationalize UNIX System V. `nawk` provides other enhancements in addition to 8-bit support, such as the ability to define functions.

In Release 4.0, awk is provided as oawk (old awk) and nawk. By default, awk is linked to oawk.

For more information about nawk, see the nawk(1) manual page.

Improved Recovery of Files from cpio Archives

Release 3.1 provided a procedure for skipping over bad blocks in a cpio archive file, in the event that errors are encountered while restoring a file from floppy disk using sysadm or cpio. The procedure allows users to continue the restore with the next file, and greatly reduces the amount of data that will be lost.

For more information, see the cpio(1) manual page.

Incremental Backup of Nested Files

A procedure was implemented for the first time in Release 3.1 to permit incremental backup of a nested file system to floppy disk (a nested file system is a file system that is not mounted directly under root).

In Release 4.0, a number of sophisticated options have been added to the backup and restore facilities. For information, see Chapter 5 of this guide or the *System Administrator's Guide*.

Swapping the User Area

The UNIX system maintains an internal data structure for each process called the user area (or u-block). Older UNIX systems keep this data locked in main memory. In Release 3.1, UNIX System V was modified so that the u-block for users whose processes are not running can be swapped out of main memory, thus freeing more system memory for the use of user programs.

Smaller and Faster curses

Release 3.1 provided changes in the `curses` library that increase efficiency of applications. Recompiling and relinking an application package with the Release 3.1 `curses` library results in an application that uses significantly less memory than it did in earlier releases, with faster execution.

In Release 4.0, `curses` is packaged with the C Software Development System, Issue 5.

Release 3.2 Features

One of the great benefits of the UNIX system when it was first developed was the ease with which the researchers working on a project could share data. This benefit was achieved, to some extent, by minimizing security barriers in the system. The UNIX system's multi-user capability is no less valuable today, but the system has now proliferated to environments in which protecting data is as important as sharing it. In response to this realization, features were added to UNIX System V in Release 3.2 to improve system security.

In addition to security features, Release 3.2 added enhancements to UNIX System V in the form of new utilities packages: the 2K File System Utilities, FACE, and FMLI.

Enhanced System Security

The following features were added to UNIX System V in Release 3.2 to improve system security. These security enhancements remain unchanged in Release 4.0.

- `lastlogin` Time

  To enhance security, the time that a user last logged in is displayed each time that user logs in.

- `loginlog` File Capability

  In Release 3.2, the system was modified to record unsuccessful login attempts in a file. If the file `/var/adm/loginlog` exists, any five consecutive unsuccessful login attempts will be logged there. If, however, a user has fewer than five unsuccessful attempts, the unsuccessful attempts are not logged in the file. (If, for example, the user logs in successfully on the fifth attempt, the four unsuccessful attempts are not recorded.)

  In Release 4.0, the `loginlog` file resides in `/var/adm`.

- Sticky Bit

  Because public directories such as `/tmp` and `/var/tmp` are writable by everyone, anyone could remove files from them. This situation posed a serious problem to the integrity of files contained in those directories, as

well as to the overall security of the system. Beginning with Release 3.2, the sticky bit on a directory is used to restrict the removal of files within that directory so that only the owner can remove the files. Without the sticky bit, the standard UNIX system semantics for object removal are followed. The Release 3.2 and Release 4.0 installation media set the sticky bit for the public directories `/tmp` and `/var/tmp`.

- Shadow Password File

  Previously, encrypted user passwords were stored in the password file (`/etc/passwd`), which was readable by all users. Beginning with Release 3.2, encrypted passwords and their attributes (such as aging information) have been moved to an access-restricted file called the shadow password file (`/etc/shadow`). The shadow password file is readable only by its owner (`root`, by default).

  In Release 3.2, migration to the shadow password scheme was optional, and both single- and dual-password file schemes were supported. In Release 4.0, use of the shadow password file is mandatory. When you install Release 4.0 on your system, your `/etc/passwd` file is converted automatically, and an `/etc/shadow` file is added to the system. (The `/etc/passwd` file still exists, but it doesn't hold the encrypted password.)

- Enhanced Shell

  In UNIX System V there are programs that change a user's effective access privileges. For security purposes, an enhanced shell was provided in Release 3.2 that resets the effective user or group ID (and possibly both) to the real ID. This occurs when the effective user ID is less than 100 (and the effective group ID is not equal to 1). Any application whose effective user and group IDs are greater than 99 is not affected by this enhancement.

  In Release 3.2, two versions of the shell were provided, with the older version provided for compatibility with applications that might not work correctly with the enhanced version. The default version for Release 3.2 was the enhanced `/usr/bin/sh`.

Release 4.0 supports four shells. The default is still the enhanced `/usr/bin/sh`. Alternative shells are the C shell, the Korn shell, and the job control shell, none of which supports the security features described above; however, all support some security features. (The older version of `/usr/bin/sh`, supported in Release 3.2 for compatibility with earlier releases, is not supported in Release 4.0.)

For information about the shells supported in Release 4.0, see "UNIX System V Release 4.0 Features" later in this guide.

2K File System Utilities

Prior to Release 4.0, the `root` and `/usr` file systems were delivered as 1K file systems, as a logical block size of 1K provides a good balance between performance and disk space usage. In Release 3.2, an administrator could choose to make a user file system a 2K file system; if the file system contains large files, 2048 byte blocks provide improved performance for applications using the file system.

In Release 4.0, the 2K file system is in the UNIX System V base product as one of several supported UNIX System V file system types. When Release 4.0 is installed, the system prompts the administrator to designate each of the standard file systems as a particular type—either as a traditional UNIX System V file system type of 0.5K, 1K, or 2K), or as a UFS file system (a new file system type that stores data in blocks as large as 8K). The default is the UNIX System V 2K file system.

For more information about file system types and Virtual File System (the architecture that makes it possible for different file system types to coexist in UNIX System V), see Chapter 5 of this guide.

Framed Access Command Environment

Release 3.2 introduced a character-based user interface for ASCII terminals called Framed Access Command Environment (FACE). Designed to present the UNIX system environment in a user-friendly manner, FACE allows a user to see the UNIX system through a world of windows, or "frames," containing menus and forms.

In Release 4.0, FACE has been enhanced to be more consistent with the version of FACE developed for UNIX System V/386 Release 3.2, and new tools have been added that make it easier to use FACE in application programs.

Form and Menu Language Interpreter

Release 3.2 introduced the Form and Menu Language Interpreter (FMLI)—a high-level language interpreter that allows developers to write user-friendly interfaces to applications that run on ASCII terminals.

In Release 4.0, FMLI includes many extensions to the Form and Menu Language, including a way to interrupt executables, a conditional statement (if-then-else), new built-in functions `test` and `expr`, and other improvements that give FMLI programmers more control over the appearance and behavior of their application interface.

Enhanced curses

The UNIX System `curses` screen management library was improved in Release 3.2 to support color text on terminals capable of displaying it. A default table of eight colors can be modified or expanded.

In Release 4.0, `curses` is packaged with the C Software Development System, Issue 5.

5. UNIX SYSTEM V RELEASE 4

5. UNIX SYSTEM V RELEASE 4

5 UNIX System V Release 4

Introduction

The goal of UNIX System V Release 4.0 is to unify important UNIX system variants into one, full-featured product that conforms to industry-defined standards for the UNIX system.

Where standards have been defined by industry-wide standards bodies, such as the IEEE P1003 POSIX Committee, UNIX System V conforms. Where standards have yet to be defined, UNIX System V incorporates *de facto* standards introduced by a number of different UNIX systems. These *de facto* standards represent the most popular features of 4.2 and 4.3 BSD, SunOS, and XENIX, as well as UNIX System V. In addition, the System V Interface Definition (SVID), AT&T's published standard for UNIX System V, has been revised and extended, submitted for industry-review, and reissued in concert with Release 4.0.

To meet the goal of unification and standardization, an extensive redesign of some aspects of the UNIX system was necessary; for example, the traditional UNIX file system becomes one of many supported file system types. Despite the scope of its changes, however, Release 4.0 provides a high degree of compatibility with previous UNIX System V releases and applications.

Incorporating features introduced in other UNIX systems extends the capabilities of UNIX System V in many areas, especially in the area of networking. For the first time in Release 4.0, UNIX System V offers full support for networking via a STREAMS-based implementation of the DARPA Internet protocol suite (TCP/IP)—a family of network protocols that has been included in BSD releases in the past.

Unifying UNIX system variants and conforming to standards does not preclude introducing new technologies and enhanced capabilities. For example, Release 4.0 introduces features that allow the UNIX system to be more useful in real-time processing environments.

The discussion of Release 4.0 that follows is not simply a list of new features. Instead, it describes various functional areas of UNIX System V as they existed prior to Release 4.0 and as they exist *in* Release 4.0. The discussion covers the following topics:

- The Command Set
- The Command Interface (the Shell)

- File Operations
- The File System
- The Directory Tree
- Input/Output
- Memory Management
- System Access
- Process Management
- System Administration and Maintenance
- Networking
- Character-Based User Interfaces
- Graphical User Interface
- Internationalization

Descriptions of new features are integrated into the discussion of each functional area. For a description of Release 4.0 organized by feature, see the "Product Overview" in the *Product Overview and Master Index*.

The Command Set

One of the goals of Release 4.0 is to incorporate the most popular BSD and XENIX commands into the UNIX System V command set. By merging the commands of the major UNIX operating system variants, Release 4.0 takes a significant step toward providing the UNIX system user with a single, consistent command set.

Merging the command sets of different systems posed a number of problems, however. In some cases, two different commands had the same name. In other cases, option or argument usage conflicted with existing standards, such as the SVID. In general, the decision was made in Release 4.0 to change command names and options when necessary, rather than sacrifice functionality or risk introducing confusion. Commands that offered the same or similar functionality have been merged into a single command.

Some commands from the UNIX system variants were not merged into the Release 4.0 command set. Many of these commands have been placed in a compatibility package. Users can access them if they wish by including the compatibility "bin" in their paths; however, users who do not need the commands are not required to install them.

The following tables list the commands from BSD, SunOS, and XENIX that have been added to or merged with the UNIX System V command set. For information about specific commands, see the manual pages in the Release 4.0 reference manuals.

BSD Commands Added to UNIX System V

arp(1M)	last(1)	rsh(1)
atq(1)	more(1)	rshd(1M)
atrm(1)	mountd(1M)	ruptime(1)
automount(1M)	named(1M)	rusers(1)
biod(1M)	netstat(1M)	rusersd(1M)
bootparamd(1M)	nfsd(1M)	rwall(1)
chkey(1)	nslookup(1M)	rwalld(1M)
clear(1)	page(1)	rwho(1)
compress(1)	ping(1M)	rwhod(1M)
csh(1)	quot(1M)	script(1)
ctags(1)	quota(1M)	showmount(1M)
dump(1)	quotacheck(1M)	spray(1)
dumpfs(1M)	quotaoff [quotaon(1M)]	sprayd(1M)
edquota(1M)	quotaon(1M)	statd(1M)
exportfs(1M)	rarpd(1M)	strings(1)
finger(1)	rcp(1)	telnet(1)
fingerd(1M)	rcpbind(1M)	telnetd(1M)
fmt(1)	rdate(1M)	tftp(1)
fold(1)	rdump [ufsdump(1M)]	tftpd(1M)
ftp(1)	repquota(1M)	timed(1M)
ftpd(1M)	restore(1M)	trpt(1M)
gcore(1)	rexecd(1M)	tunefs(1M)
gettable(1M)	rksh [ksh(1)]	uncompress [compress(1)]
head(1)	rlogin(1)	uudecode [uuencode(1C)]
htable(1M)	rlogind(1M)	uuencode(1C)
ifconfig(1M)	route(1M)	whois(1)
inetd(1M)	routed(1M)	zcat(1) [compress(1)]
keyenvoy(1M)	rpcgen(1)	zdump(1M)
keylogin(1)	rpcinfo(1M)	zic(1M)
keyserv(1M)	rquotad(1M)	
ksh(1)	rrestore [ufsrestore(1M)]	

BSD Commands Merged with UNIX System V Commands

admin(1)	fsck(1M)	rmdel(1)
at(1)	get(1)	sccsdiff(1)
bc(1)	id(1M)	sed(1)
chgrp(1)	init(1M)	sh(1)
chmod(1)	join(1)	sort(1)
chown(1)	kill(1)	spline(1G)
cp(1)	ln(1)	split(1)
cpio(1)	login(1)	stty(1)
crontab(1)	ls(1)	tail(1)
date(1)	mach [uname(1)]	tar(1)
dc(1)	mkfs(1M)	test(1)
delta(1)	mount(1M)	touch(1)
devnm(1M)	mv(1)	umount(1M)
df(1M)	ncheck(1M)	val(1)
diff(1)	od(1)	vi(1)
diff3(1)	pr(1)	vmstat [sar(1)]
echo(1)	prs(1)	w [whodo(1M)]
ed(1)	pstat [swap(1M)]	wall(1M)
find(1)	rm(1)	write(1)

XENIX Commands Merged with UNIX System V/BSD Commands

crash(1M)	grep(1)	passwd(1)
cron(1M)	init(1M)	su(1M)
echo(1)	ipcs(1)	sulogin(1)
file(1)	login(1)	tar(1)
fsck(1M)	ls(1)	
fsdb(1M)	mknod(1M)	

The Command Interface

For the first time in Release 4.0, UNIX System V supports four command line interfaces, or shells. These are

- The UNIX System V shell (`sh`)
- The C shell (`csh`)
- The Korn shell (`ksh`)
- The job control shell (`jsh`)

The UNIX System V shell (also known as the Bourne shell) is the default shell in Release 4.0. The UNIX System V shell is documented in the SVID and provides the greatest portability for shell programmers.

The C shell (developed at the University of California at Berkeley) is a popular command interface and programming language. Its popularity stems from its command re-execution and editing facilities.

The Korn shell is supported in Release 4.0 because it provides the basic functionality of the Bourne shell, but with expanded capabilities as an interactive interface. It maintains a command history file and supports editor interfaces for searching, modifying, and re-executing commands.

The job control shell (`jsh`) provides an implementation of job control that conforms to the POSIX standard as defined in POSIX P1003.1. It allows users to stop and restart jobs in the background and foreground, and to move jobs back and forth between the foreground and the background.

For more information, see the *User's Guide* and the `sh`(1), `csh`(1), and `ksh`(1) manual pages.

File Operations

File operations have been improved in Release 4.0 for robustness and ease of use and to unify BSD, XENIX, and UNIX System V file manipulation interfaces. Release 4.0 incorporates POSIX P1003.1 as well as popular BSD and XENIX file manipulation system calls.

For detailed information about the features described in this section, see the *Programmer's Guide: System Services and Application Packaging Tools.*

Dynamic Adjustment of the Number of Open Files

As the applications that UNIX System V runs grow in complexity and size, certain traditional limits become obstacles for programmers. One such limit is the number of file descriptors that a process may have active at any given time. Previous to Release 3, UNIX System V had a hard-coded limit of 20 open file descriptors per process. Although sufficient for most programs, this limit complicates the writing of some programs, such as database managers and network daemons.

In Release 3, the number of files that a single process could have open simultaneously was adjustable on a system-wide basis with the NOFILE parameter. However, NOFILE set a fixed upper bound that could not be exceeded by any process.

In Release 4.0, NOFILE has been removed; the number of files that may be opened is dynamically tunable for each process and has no inherent upper limit. This feature makes it possible for a program to monitor many files, devices, or network ports simultaneously.

Memory-Mapped Files

The traditional method of manipulating files and devices makes resources accessible to user processes only by means of explicit system calls. Accessing a memory location in a process and accessing a file or device are completely different operations.

This complicates the operation of some programs and makes others impossible to write without writing kernel code. To address the problem, a file mapping mechanism has been introduced that allows programs to access files and devices

as ranges of bytes within a process's virtual address space. Once a file is mapped into the process's address space, its contents can be accessed as memory locations in the process.

Memory-mapped files are a by-product of the new UNIX System V Virtual Memory (VM) architecture. For more information about VM and memory-mapped files, see the discussion of memory management later in this section.

POSIX, BSD, and XENIX File Operations

Release 4.0 supports system calls that control such file operations as file renaming and file truncation as defined by the IEEE P1003 (POSIX) Standard Committee. Some of these system calls, such as `truncate` and `ftruncate`, were originally defined by BSD, then adopted by POSIX.

Release 4.0 also supports the BSD system call `fsync` for synchronizing in-memory file contents and the contents on the disk. This file synchronization mechanism is provided in addition to the UNIX System V O_SYNC mode. The `fsync` system call is provided as an alternative mechanism that avoids the performance penalty incurred by processes using the O_SYNC mode.

In addition to the Advisory and Mandatory Record Locking modes introduced in Release 2.0 and 2.1 respectively, Release 4.0 supports XENIX locking. With XENIX locking, a process can lock a file or a portion of a file with the mandatory locking semantics supported by the XENIX operating system.

For more information about POSIX conformance and BSD compatibility, see the *Standards and Conformance Guide and Reference Manual.*

The File System

In earlier UNIX systems, the file system was a static entity of fixed size and format. As the UNIX system was extended to solve new problems, the traditional file system was no longer adequate. Release 2 introduced a "convertible" file system switch that allowed a system to support both 0.5K and 1K file systems. Beginning in Release 3.2, a 2K file system was supported. In Release 4.0, Virtual File System (VFS) has been implemented as a means of generalizing further the concept of a file system, so that file systems of different types can be supported simultaneously on the system.

Virtual File System

Virtual File System (VFS) is a file system switch architecture that replaces the "convertible" 0.5K or 1K UNIX System V File System introduced in Release 2. The implementation of VFS in Release 4.0 is based on the VFS architecture first implemented in BSD UNIX systems.

The architecture provides a clearly-defined, modular interface between the file system and the rest of the UNIX system kernel and allows several different types of file systems to exist simultaneously on the system. These file systems may have widely different characteristics and internal formats; for example, the VFS architecture allows a set of remote files and a set of traditional UNIX System V files to exist on the same system. The system mandates access to the different classes of files so that user programs do not need to know what type of file is being accessed or modified.

VFS provides a file-system-type-independent interface to programs and users while allowing each particular file system to interpret these operations in its own manner. File-system-type-dependent kernel routines do the work specific to the type.

A key strength of the VFS architecture is that it allows new file system types to be defined and implemented easily. The modular nature of the architecture allows programmers to design and install new types of file systems in a clean, straight-forward manner. Configuring a new file type into the system requires approximately the same level of effort as installing a new device driver.

For more information about VFS, see the *Programmer's Guide: System Services and Application Packaging Tools*.

File System Types

The new UNIX System V file system architecture allows a wide variety of file system types to exist on the same system. UNIX System V Release 4.0 provides the following file system types:

`s5` The traditional UNIX System V file system, supporting data storage in disk blocks of 0.5K, 1K, or 2K bytes.

`ufs` An implementation of the BSD "fast file system," a file system type that stores data in disk blocks as large as 8K bytes. UFS supports all SVID-defined file operations.

`rfs` Remote File Sharing (RFS), a file sharing utility re-implemented under the Virtual File System architecture as a distributed file system type. RFS supports sharing files across a network of UNIX systems.

`nfs` Network File System (NFS), a new implementation of a distributed file system type originally developed by Sun Microsystems. NFS makes it possible for systems of different architectures running different operating systems to share text and data files across a network.

`/proc` The process file system type, which is a mechanism for accessing the address space of running processes. `/proc` is particularly useful for debuggers and similar utilities.

`fifofs` A file system type that provides common access to pipe files.

`specfs` A file system type that provides a common-code interface to all device or "special" files.

`bfs` A file system type that provides support for file-system-independent booting; a `bfs` file system contains all the programs necessary for the boot process.

For more information about file system types, see the *System Administrator's Guide.*

File-System-Independent Booting and Autoconfiguration

File-system-independent booting allows an administrator to boot the system from any device that is readable by firmware.

In earlier releases of UNIX System V, the boot-strap program assumed that the root file system was of a particular file system type. This assumption was necessary since all the bootable programs (such as the UNIX system kernel itself) resided in the root file system; the boot program needed to know the structure of the root file system so it could locate programs.

In Release 4.0, it is possible to make the root file system any of the supported file system types. Rather than coding knowledge of all possible file systems into the boot program, a new file system has been created to hold all the bootable programs necessary for the boot procedure. In this way, the boot program no longer depends on the file system type of the root file system.

When Release 4.0 is installed, a new partition called `/stand` is created to hold a file system of type `bfs` (Boot File System). This file system contains all the programs necessary for the computer's boot-strap process, including `unix`, the bootable operating system, and the system file (`/stand/system` in previous releases).

Autoconfiguration—the process of detecting hardware and software changes on powerup or reboot and generating a new bootable operating system—has been made faster for Release 4.0, and operates in the same file-system-independent manner as the boot program.

In addition, a user-level interface to system configuration, `cunix`, has been added in Release 4.0. This allows experienced administrators to create a new operating system image without powering down or restarting the system.

For more information, see the *System Administrator's Guide.*

File and File System Status System Calls

UNIX system programs use the `stat` system call to obtain detailed information about a file. The BSD `stat` structure contains two fields not in the Release 3 structure—`st_blocks` and `st_blksize`. To allow binary compatibility with Release 3 programs, the Release 4.0 `stat` structure is identical to the Release 3.2 `stat` structure. BSD programs requiring the two additional fields need to be relinked with a compatibility library.

To obtain information about a file system, Release 3 introduced the `statfs` system call and structure. With the advent of the Virtual File System, the `statfs` call is obsolete. Programs should be modified to use the Release 4.0 `statvfs` system call.

The `statfs` call is provided for compatibility with existing Release 3 programs.

For more information about `stat`, see the `stat`(2) manual page.

File Linking

Early UNIX systems allowed a single file to be assigned more than one name through a "linking" mechanism; however, this mechanism was limited. Directory files could not have more than one name, and all names associated with a file had to reside within a single physical file system.

Symbolic links overcome these limitations. A symbolic link is a file that contains the pathname of another file. References to the symbolic link are converted by the UNIX system kernel into references to the target file.

Because symbolic links allow directory files to be linked, the logical structure of a system's file tree can be rearranged without changing the physical locations of files. Because symbolic links allow links to a physical file to reside in a different physical file system, names can be linked across file systems that reside on different computers on a network—allowing a computer to create a logical directory tree that includes directories and files that physically reside on many different computers.

Release 4.0 supports both traditional and symbolic links. The `ln` command interprets the `-s` option as a request to make a symbolic link. In addition, Release 4.0 supports the system calls `symlink`, `readlink`, `lstat`, and `lchown` for manipulating symbolic links.

For more information about symbolic links, see the *Programmer's Guide: System Services and Application Packaging Tools*.

The Directory Tree

In Release 4.0, the layout of the directory tree has been rearranged to make it easier to share resources across a network. Within the directory layout, the `/dev` directory has been restructured so that its size is more manageable.

The Directory Layout

The UNIX system directory tree was created in the days of stand-alone, disk-based minicomputers. In Release 4.0, the physical organization of files and directories in the UNIX system has been changed to facilitate sharing files in a network environment. Files of different types have been separated into different subtrees of the UNIX system directory.

Release 4.0 divides the system into the following file systems:

`root` which contains all the files necessary for booting the system.

`/usr` which has been reorganized in Release 4.0 to contain shareable files that are static over the life of the system; system-modified files previously located in `/usr` have been moved to `/var`.

`/home` which contains the home directories and files of the system's users.

`/var` which contains files and directories whose contents change over the life of the local system, such as system log files.

In addition to these standard file systems, Release 4.0 creates a `/stand` partition, where it stores all the files needed for file-system-independent booting.

The file types defined in Release 4.0 are

- Machine-private files
- Architecture-dependent files
- Architecture-independent files

Machine-private files are those files that cannot or should not be shared with other machines, regardless of CPU type. Examples of machine-private files include `/sbin/init.d` scripts for machine-specific boot procedures, or accounting logs. The root file system in Release 4.0 contains the machine-private files for the system.

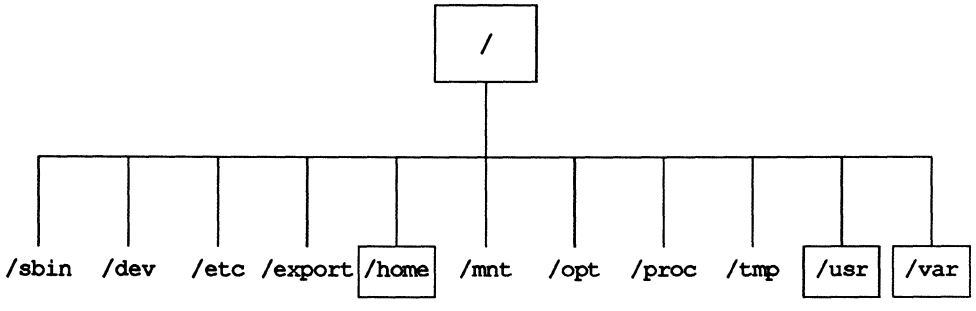
Architecture-dependent files are files that are shareable across a network between machines of the same CPU type. Examples of architecture-dependent files are binary executables. The `/usr` file system in Release 4.0 contains the architecture-dependent files.

Architecture-independent files are files that can be shared across a network regardless of CPU type. Examples of architecture-independent files include ASCII databases and on-line manual pages. The `/usr/share` directory in Release 4.0 contains architecture-independent files.

To make it easier for pre-4.0 users to adapt to the new layout, Release 4 preserves the Release 3 name space and provides symbolic links so that system directories and files can be accessed using pre-4.0 path names. For information about symbolic links, see "File Linking" earlier in this chapter.

The Root File System

In Release 4.0, the root file system is structured as follows:

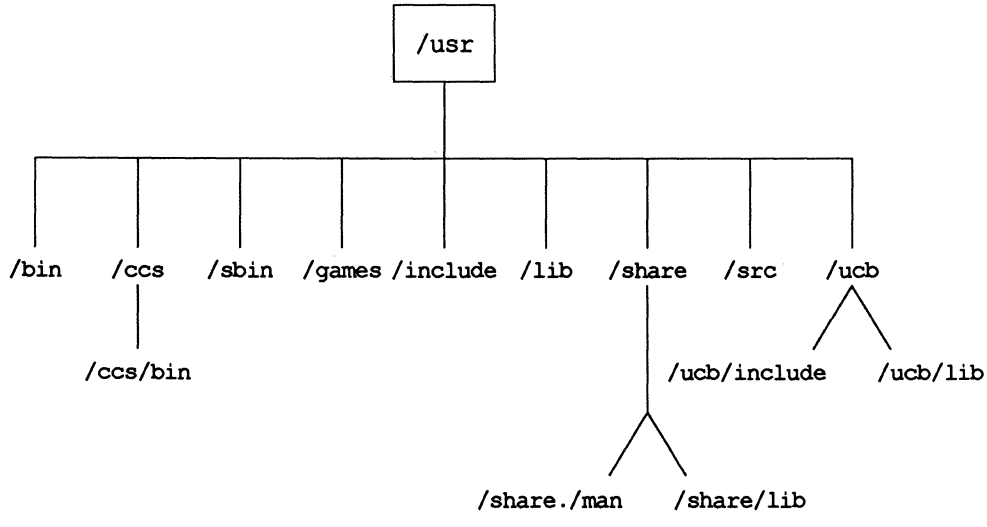


`/sbin`	Essential executables for administration and operations
`/dev`	Character and block special files
`/etc`	Machine-specific administration configuration files and system administration databases (`/etc` does not contain executables)

/export	The default root of the exported file system tree
/home	Root of a standard file system for user directories
/mnt	The default temporary mount point for file systems
/opt	The root of a subtree for add-on applications packages
/proc	The root of the process file system
/tmp	System generated temporary files
/usr	Root of a standard file system containing static, shareable files
/var	Root of a standard file system for varying files

The /usr File System

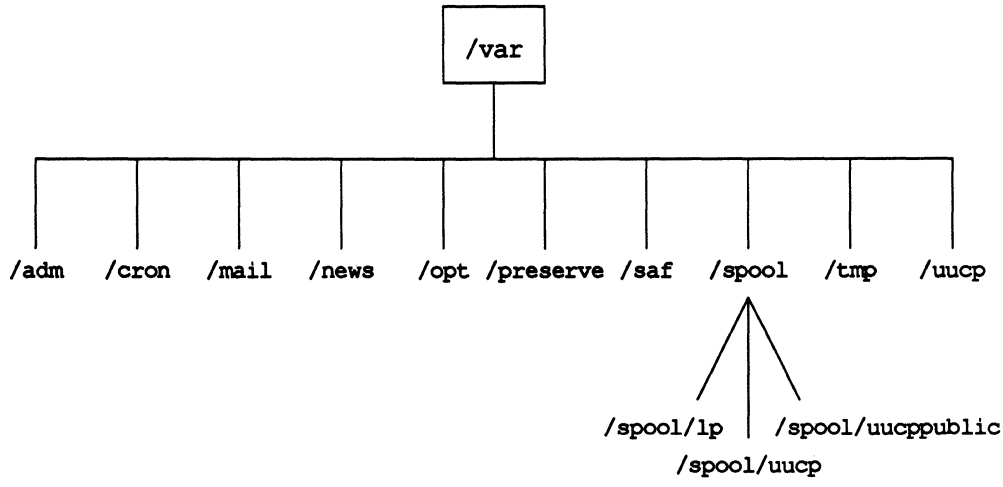
Release 4.0 restructures the /usr file system as follows:



/usr/bin	A directory containing the majority of system utilities
/usr/ccs	The C Compilation System
/usr/ccs/bin	The C Compilation System utilities
/usr/sbin	Executables for system administration
/usr/games	Game binaries and data (if installed)
/usr/include	Program header files
/usr/lib	Program libraries and architecture-dependent databases
/usr/share	Architecture-independent shareable files
/usr/share/man	On-line manual pages (if installed)
/usr/share/lib	Architecture-independent databases (e.g., ASCII files)
/usr/src	Source code for utilities and libraries (source-code licensees only)
/usr/ucb	BSD compatibility package binaries
/usr/ucb/include	BSD compatibility header files
/usr/ucb/lib	BSD compatibility libraries

The /var File System

The /var file system is structured as follows:



/var/adm	System logging and accounting files
/var/cron	Directory containing a log for the cron command
/var/mail	User mail files
/var/news	Common interest messages
/var/options	Directory containing a file that identifies each utility installed on the system
/var/preserve	Backup files for vi and ex
/var/saf	Service Access Facility logging and accounting files
/var/spool	Directories for spool temporary files
/var/spool/lp	A line printer spooler temporary directory
/var/spool/uucp	Queued uucp jobs

/var/spool/uucppublic	Files deposited by uucp
/var/tmp	A directory for temporary files
/var/uucp	uucp log and status files

/dev Restructuring

The /dev directory, the directory that holds device-specific files, has been restructured in Release 4.0. As computers are able to support larger numbers of terminals and pseudo-terminals, the number of terminal-related devices in the /dev directory quickly becomes unmanageable. To alleviate this problem, subdirectories have been created under /dev, and the ttyname library routine has been enhanced to search subdirectories. Terminal-related devices are now located in /dev/term. Pseudo-terminals reside in a subdirectory called /dev/pts. xt device files used by layers now reside in the /dev/xt directory, and shell layers device files used by shl reside in the /dev/sxt directory.

With the enhanced ttyname, new subdirectories may be created under /dev, and other peripheral ports devices need not be located in /dev/term.

ttyname has been enhanced to read a configuration file called /etc/ttysrch. /etc/ttysrch is a file that the system administrator creates if he or she wishes. It specifies the names of the directories that should be searched and the order in which they should be searched. Each line of the file is a directory name, including the full path. ttyname searches the directories in the order in which they are listed in the file. If ttysrch is not created, ttyname uses a default search list that consists of /dev/term, /dev/pts, and /dev/xt. If a match is not found, it then searches through the rest of the /dev directory and its other subdirectories.

Pre-4.0 add-on packages that are dependent on the pre-4.0 /dev structure continue to work in 4.0, provided an administrator runs a script to create links from the new structure to the pre-4.0 structure. The script /usr/sbin/lnttys is provided to create the necessary links. When an administrator invokes the script, links are created from /dev/term/*X* to /dev/tty*X*.

To preserve compatibility with add-on packages, it may also be necessary to create or edit the ttysrch file so that it searches the /dev directory first before looking in the subdirectories. This will ensure that ttyname returns the old-style name rather than the new style.

The new `/dev` structure does not preclude subdirectories from being created to contain other related devices; however, it is strongly recommended that only one level of subdirectory exist under `/dev`.

For more information, see the *System Administrator's Guide* and the `ttyname`(3C) and `ttysrch`(4) manual pages.

Input/Output

In the UNIX system I/O subsystem, new kernel software is required to interface with each new device that is added to the system. In many cases, there may be a great deal of overlap between existing device drivers and new drivers, but there is no convenient way for driver writers to share common code.

In Release 3.0, UNIX System V introduced STREAMS, an I/O mechanism that overcomes some limitations of traditional UNIX system I/O by providing greater modularity and flexibility of device drivers and protocols. Release 4.0 extends the use of the STREAMS mechanism in UNIX System V and introduces the Device Driver Interface (DDI) and the Driver-Kernel Interface (DKI), interfaces between a device driver and the kernel that make it easier to port driver source code.

STREAMS

Release 3.0 introduced the STREAMS mechanism as a framework for UNIX system character I/O. The STREAMS mechanism allows the structuring of kernel software in a modular manner.

STREAMS was introduced initially as a framework for implementing network protocols. By defining a high-level set of network interfaces, the actual device driver (and therefore the network protocol and media) are not directly visible to applications programmers. This allows the creation of networking applications that do not depend on a particular network or protocol to operate.

In Release 4.0, the entire terminal (tty) subsystem in the kernel has been rewritten to use the STREAMS mechanism. The move to a STREAMS implementation increases the modularity of the tty subsystem and provides a more flexible framework for future enchancements. Pseudo-ttys (ptys) have been implemented under STREAMS as well.

To unify the interfaces that programs use to communicate with character devices and processes, Release 4.0 re-implements pipes using the STREAMS mechanisms. Release 4.0 provides compatibility with programs written prior to the introduction of STREAMS-based pipes; existing programs that use pipes continue to operate as they did previously.

For more information about the STREAMS mechanisms in Release 4.0, see the *Programmer's Guide: STREAMS*.

Device-Kernel Interface/Device Driver Interface

In Release 4.0, UNIX System V supports the Device-Kernel Interface (DKI), an interface between the UNIX kernel and device driver software. DKI makes it easier to port driver code across implementations of Release 4.0 for different hardware; if the driver writer conforms to the DKI where possible, the portability of the driver to other UNIX System V Release 4.0 implementations is greatly enhanced.

Release 4.0 also supports the Device Driver Interface (DDI), a superset of the DKI that is specific to the implementation of UNIX System V for the AT&T 3B2 line of computers. The DDI enhances driver binary compatibility across releases of UNIX System V for the 3B2 computers.

See the *Device Driver Interface/Driver-Kernel Interface (DDI/DKI) Reference Manual* for more information.

Memory Management

Release 4.0 incorporates a third-generation memory management architecture based on the Virtual Memory (VM) architecture previously implemented in SunOS. VM replaces both the original UNIX system swapping architecture and the REGIONS demand-paged virtual memory architecture introduced in Release 2.1.

The benefits of a demand-paged virtual memory implementation are efficient use of the system's main memory and the capability to execute programs much larger than the physical memory provided by the system. The Release 4.0 VM architecture provides these same benefits, plus the benefits listed below.

- Mapped files (application programmers)

  A by-product of the VM architecture is an entirely new style of file I/O for user programs called "mapped files." This new set of capabilities, provided by the *mmap* family of routines, allows a file to be mapped explicitly into the address space of a user program, where the file can be manipulated as if it were an array in primary memory. These new capabilities make user programs easier to write and more efficient to run.

- Shared memory (application programmers)

  The mapped files provided by VM can be considered a form of shared memory in Release 4.0. If several processes map a file simultaneously, the system maps the same copy of a file into the memory space of all processes.

  The traditional UNIX System V shared memory facility, provided by the *shmat* family of routines, continues to be available in Release 4.0. New in Release 4.0 is support for XENIX shared memory semantics, including full source-code compatibility with older XENIX systems.

- Flexible use of disk space for swapping (administrators)

  Older systems normally swap memory pages to a formatted physical device partition of a fixed size. In Release 4.0, UNIX System V can also swap pages of data from main memory to an ordinary file on one of the system's disks. Swapping to a file accommodates the needs of diskless systems and allows for the more efficient use of disk space.

- Portable implementation (system providers)

  The Release 4.0 implementation of VM isolates all hardware-dependent portions of the memory management subsystem in one block of C language source code. The remainder of the code is portable across different hardware and system architectures. Most of the hardware-dependent code comprises a well-defined Memory Management Unit (MMU) interface. This interface allows the architecture to be implemented on top of different MMU hardware.

  Release 4.0 provides a "multi-level store" implementation of VM that allows a system to address extremely large ranges of virtual address space using only a 32-bit hardware address space—the industry standard for central processors and hardware memory management units.

For more information about VM, see the *Programmer's Guide: System Services and Application Packaging Tools.*

System Access

The programs involved in allowing users to log in to a UNIX system depend in part on the physical aspects of the specific connection. Pre-4.0 UNIX systems provide two types of external access points. If the user is at a terminal with a connection to the computer via a tty line, the `getty` program provides access to the system. If the user is attempting to log in from another machine across a local area network, a network connection agent—such as `listen`—monitors a set of network addresses to which the remote user's `cu` or `rlogin` programs transmit. A problem with this approach to system access is that there are three facilities, all with different interfaces. Another problem is that there is no easy way to disable all external access to the system on demand.

To address these problems, Release 4.0 introduces the Service Access Facility (SAF)—a facility that acts as an umbrella over the system's external access points and provides a consistent access mechanism. The SAF allows services to be administered through a consistent framework of commands and files, whether the service request comes via a tty line or across a network.

Port Monitors

In Release 4.0, a port monitor is a process under the control of the Service Access Facility that detects activity on a port, or on more than one port of the same type (tty lines or network addresses). Release 4.0 provides three port monitors under the SAF umbrella—two network port monitors, `listen` and `inetd`, and a serial port monitor called `ttymon`. `listen`, introduced as the Listener in Release 3.2, is a general purpose network listener that provides connection-oriented service in a protocol-independent manner. `inetd` is the DARPA port monitor. It provides connection and connectionless service on a TCP/IP network.

In Release 4.0, the `ttymon` port monitor replaces the `getty` program. A single `ttymon` process can monitor many serial ports and eliminate the need for a `getty` process for each port. `ttymon` makes use of two important Release 4.0 features: the ability to poll STREAMS-based ttys, and the virtually unlimited number of file descriptors that a privileged process may have open.

The `getty` program no longer exists in Release 4.0. A symbolic link from `getty` to `ttymon` is provided to maintain compatibility with applications that explicitly invoke or configure a `getty`. These applications, however, should be modified to use the administration interfaces available under the Service Access Facility.

In addition to the standard port monitors provided by Release 4.0, the SAF architecture allows new port monitors to be installed by users and applications.

For more information about port monitors, see the *System Administrator's Guide* and the `ttymon`(1M) and `listen`(1M) manual pages.

Service Access Controller

The Service Access Controller (SAC) manages port monitors. It provides the system administrator with the `sacadm` command, which allows the administrator to add and remove, start and stop, and enable and disable a port monitor.

For more information, see the *System Administrator's Guide* and the `sacadm`(1M) manual page.

Process Management

Release 4.0 introduces a number of new features and enhancements to UNIX System V in the area of process management. These features include BSD job control, Expanded Fundamental Types (EFT), an enhanced signal interface, and features that provide some support of real-time applications.

Job Control

Job control is a popular feature of the BSD operating system and an optional part of the IEEE P1003.1 POSIX standard. Job control allows a user to stop and later resume a job, whether it is executing in the foreground or the background. Job control also allows a user to move jobs back and forth between the background and the foreground.

With job control a user can

- stop a foreground job in order to perform a more pressing task
- put a foreground job in the background
- stop a job to satisfy a need of the job, such as looking up data for input or changing the name of an input file to match what was misspelled on the command line.

Job control capabilities are available through an optional shell called the job control shell (`jsh`). For more information about job control, see the *User's Guide*. Processes in the UNIX system receive asynchronous notification via the signal mechanism. The notification is asynchronous because a signal can arrive at any time during the execution of a process.

For information about the job control shell, see the sh(1) manual page.

Expanded Fundamental Types

UNIX System V Release 4.0 supports the expansion in size of certain data types, such as user ID (uid), process ID (pid), and device ID. This feature, known as Expanded Fundamental Types (EFT), makes it possible to remove the artificial constraints imposed on these data types by the UNIX operating system's original hardware implementation.

Enhanced Signals

Over the years, many variations of the original UNIX system signal mechanism have been created to extend the usefulness of signals and to increase their robustness. Recently, POSIX P1003.1 has consolidated these signal interfaces into a single standard.

The POSIX signal interface features:

- the ability to manipulate a set of signals
- the ability to block and unblock signals
- the ability to examine pending signals
- job control support

In addition to the POSIX signal interface, Release 4.0 supports the the complete pre-3.0 interface (`signal`) and the Release 3.0 interface (`sigset`). All three signal interfaces may be mixed in a single program.

For more information about the POSIX signal interface, see the `signal`(5) manual page.

Real-Time Support

Historically, the UNIX system has been a general purpose timesharing system. Today, however, there is growing demand for real-time applications support. To address that demand, Release 4.0 introduces a new process-scheduler architecture and high-resolution timing services.

For more information about real-time support, see the *Programmer's Guide: System Services and Application Packaging Tools.*

User-Controlled Process Scheduler

A process scheduler is part of the system kernel that determines what processes will run, when, and for how long. The Release 4.0 architecture supports several different schedulers simultaneously. The release supplies both the traditional timesharing scheduling policy and the new real-time scheduling policy. Each process can have its own scheduler properties, and they can be changed by the process while it is running.

The traditional UNIX System V scheduler policy manages processes in a traditional manner: it dynamically adjusts time-sharing process priorities in an attempt to give good response to all interactive processes.

The real-time scheduler policy never changes a process's priority except as a result of an explicit user request. Moreover, all real-time processes run before any other processes. An application can perform its time-critical tasks and be assured that it will always get priority over all other processes.

Also to support real-time processing, Release 4.0 provides new pre-emption points in the kernel—points at which the scheduler may switch control of the CPU from one process to another. The additional pre-emption points improve system response time for high-priority processes.

Using the new scheduling facilities, an application can guarantee fast, deterministic response to its critical processes, with a resolution of milliseconds rather than seconds.

High-Resolution Timers

For applications that deal with very short intervals, Release 4.0 provides BSD timing services that give microsecond resolution. These services include alarms, interval timers, and a time-of-day clock.

System Administration and Maintenance

Release 4.0 introduces a number of improvements and enhancements to UNIX System V operations, administration, and maintenance with the goal both to simplify administration and maintenance operations and to provide new capabilities.

For detailed information about the features described in this section, see the *System Administrator's Guide.*

Backup and Restore Operations

Backup and restore procedures have been made hardware-independent in Release 4.0. They now support multiple bus architectures and multiple destination types, such as tapes, floppy diskettes, and hard disks. Multiple commands used in the backup procedure have been integrated into a single backup service. The key features of the backup and restore service include

- a backup history log
- on-line backups
- automated backup initiation
- mechanized restore requests

Software Installation

In Release 4.0, installation routines have been made consistent across software packages, releases, and machines. Also provided in Release 4.0 are tools and guidelines for developing add-on packages that take advantage of the standard software installation script, as well as the menu interface.

System Administration Menus

Release 4.0 introduces enhancements to the System Administration Menus (accessed through the `sysadm` command) that make the interface simpler in design and easier to use.

Networking

Early networking capabilities in the UNIX system consisted of the uucp networking package, included in the Basic Networking Utilities (BNU) package in UNIX System V. uucp networks provide queued point-to-point communication between computers over standard telephone lines. A uucp client process queries a database for address and routing information, in this case, a telephone number, and calls a remote system.

In the 1970s, the Advanced Research Project Agency (now the Defense Advanced Research Project Agency, DARPA) developed TCP/IP, sometimes called the DARPA Internet protocol suite. TCP/IP was designed to be a set of mid-level communications protocols for use with the ARPANET wide-area packet-switching data communications network (a network that grew to include hundreds of nodes throughout the United States). In the late 1970s, the University of California at Berkeley included an implementation of TCP/IP in its UNIX Software Distribution.

Building on the BSD innovations in UNIX system networking, SunOS introduced an implementation of a Remote Procedure Call (RPC) facility, along with a file distribution service called Network File System (NFS). RPC is a mechanism that allows a local process to invoke a procedure residing on a remote system. NFS is an application that provides transparent file sharing among computers of different architectures.

The thrust in recent years has been to develop interfaces that allow network applications to be independent of network media and protocols. In 1982, BSD introduced a networking interface called Sockets. In Release 3, UNIX System V introduced STREAMS—another solution to the same problems Sockets sought to address. STREAMS is a mechanism that supports modular development of network protocols and device drivers.

Release 3 also introduced the Transport Level Interface (TLI), a protocol-independent programming interface to networking protocols, and the listener, a program that listens for requests for service from remote machines. Also appearing for the first time in Release 3 was Remote File Sharing (RFS), a network application that allows systems to share files transparently across a network connection.

Release 4.0 extends AT&T's commitment to networking, as well as to the unification of the various commercial implementations of the UNIX system, by incorporating many of the networking features of BSD and SunOS into UNIX System V. These features are:

- Sockets compatibility library
- `inetd`
- Network File System (NFS)
- Remote Procedure Call (RPC)
- External Data Representation (XDR)
- TCP/IP (networking protocols)

In addition to supporting BSD and SunOS features, Release 4.0 introduces the following new features:

- Network Selection
- Name-to-Address Mapping
- Service Access Facility

All of the Release 4.0 networking features are described in the remainder of this section.

Sockets

Sockets is a network interface used widely in BSD systems and in derivative operating systems such as SunOS. It is functionally similar to the Transport Level Interface (TLI) provided in UNIX System V.

Sockets is provided as a compatibility library in Release 4.0 so that existing Sockets applications can migrate easily to UNIX System V. To run on UNIX System V, Sockets applications must be recompiled and relinked to the sockets library in Release 4.0.

TLI remains the SVID-defined networking interface. Programmers are encouraged to write new applications using the TLI rather than Sockets.

For more information about the Sockets procedure calls supported in Release 4.0, see the *Programmer's Guide: Networking Interfaces.*

TCP/IP Protocols and Commands

The development of TCP/IP networking began in 1969 and has grown to become the *de facto* non-proprietary standard for interconnecting computers of different types. It has been widely implemented on many classes of machines from PCs to mainframes and on both wide and local area network media.

The TCP/IP Internet package in UNIX System V Release 4.0 is a comprehensive implementation of the DARPA protocols, supporting the DARPA commands and popular BSD networking commands. It is compatible with the DARPA package implemented on BSD systems, as well as any implementation conforming to DARPA standards.

UNIX System V implements the protocol suite under the STREAMS networking architecture. This means that all user programs written to TLI run over TCP/IP without modification. It also means that system-provided networking services, such as the RFS and NFS file sharing utilities, run on the TCP/IP protocols.

For more information about the TCP/IP Internet package, see the *Network User's and Administrator's Guide.*

inetd

`inetd` is a BSD network port monitor that originated in BSD UNIX systems. A port monitor is a program that performs server-side connection management. When a connection request arrives over the network, the port monitor spawns the server and passes the network connection to it.

`inetd` handles both connection and connectionless requests from remote systems on a network using TCP or UDP protocols.

For more information about `inetd`, see the *Network User's and Administrator's Guide* and the `inetd`(1M) manual page.

Network File System

The Network File System (NFS) is a facility for sharing files in a heterogeneous environment of machines and operating systems. Sharing is accomplished by mounting a remote file system, then reading or writing files in place. Users are able to access the files they want without knowing the network address of the data. To the user, all NFS-mounted file systems look like private disks; there are no apparent differences between reading or writing a file on a local disk, and reading or writing a file on a disk in the next building.

NFS was designed as a network service, and not as a distributed operating system. It is able to support distributed applications without restricting the network to a single operating system.

For more information about Network File System, see the *Network User's and Administrator's Guide.*

Remote Procedure Call

The RPC library implements a published, industry standard protocol that can be used on many different types of computers running different operating systems. It provides a mechanism that makes it possible for the syntax and semantics of the local procedure call model to be used to invoke a process on another computer.

The RPC library uses External Data Representation (XDR) to encode data passed from one computer to another so that a computer can call a procedure on another computer running a different operating system. (The XDR library is described later in this section.)

The RPC library allows server programs to become building blocks that can be used to create more complex applications. A server might provide a service to clients by calling other servers, each of which would perform a single operation toward the accomplishment of a multi-step task. The process is much like using program modules to create larger programs, but with the added flexibility that the modules are bound dynamically at run time and can be shared and distributed.

RPC service is implemented over the Transport Level Interface (TLI), which gives it transport protocol independence and allows it to run unchanged over different networks conforming to the Transport Provider Interface.

For more information about RPC, see the *Programmer's Guide: Networking Interfaces*.

External Data Representation

Data is represented in different ways on different computers and in different programming languages. When data needs to be exchanged between two computers, these differences must be reconciled.

External Data representation (XDR) is one specification of a standard representation for data types. It is defined independently of any specific hardware, operating system, or programming language.

XDR takes care of problems with data byte ordering, data type size, and data representation by specifying what they should be. A program needs to translate between its internal representations and the XDR standards when it communicates with other computers.

Network Selection

Network Selection is a feature that helps applications select a network to use for communication. Users can specify their preference in an environment variable, NETPATH. The system administrator can set a default NETPATH for `login`, which users can override or append to as necessary. Applications have the option of using the network specified by the user in NETPATH, or selecting a network based on other application-specific criteria.

The benefit of Network Selection is that a network selection no longer needs to be embedded in the application code. This allows the application to run without change on different systems connected to different networks.

Network Selection allows a system to have a different list of networks for different applications. It also allows applications to connect to a number of different networks until it finds one that meets its service requirements and permits the connection.

For more information about Network Selection, see the *System Administrator's Guide* and the *Programmer's Guide: Networking Interfaces*.

Name-to-Address Mapping

UNIX System V Release 4.0 networking includes a name-to-address mapping mechanism that network clients can use to determine the addresses of servers in a network-independent manner. It allows clients to reach servers, even if the address on which the server is listening changes. It makes it possible for clients to be independent of networking protocols, as long as the network provides a transport-level interface. It also allows a client to reach a server through the most convenient network.

A client can identify a server by

- a service name
- the name of the host computer on which the service resides
- the name of the network to be used to reach the host.

The name-to-address mapping mechanism supports many different look-up schemes. The name-to-address daemon receives translation requests from a client and uses the `/etc/netconfig` file to obtain the name of a routine to perform the actual translations.

For more information about name-to-address mapping, see the *System Administrator's Guide*.

Service Access Facility

The Service Access Facility (SAF), described earlier in this guide, provides a uniform framework for managing external access to the system. The daemon processes (port monitors) monitor all access points to the system, including network ports, for connection requests. When a port monitor gets a connection request, it invokes the desired service. The SAF makes service access easier to manage and enhance.

For more information, see the *System Administrator's Guide*.

Character-Based User Interfaces

An earlier release of UNIX System V introduced a high-level language interpreter called Form and Menu Language Interpreter (FMLI). FMLI allows developers to write user-friendly interfaces to their applications. Release 4.0 provides extensions to the Form and Menu Language, including a way to interrupt executables, a conditional statement (if-then-else), new built-in functions `test` and `expr`, and other improvements that give FMLI programmers more control over the appearance and behavior of their application interface.

In addition, Release 4.0 provides enhancements to Framed Access Command Environment (FACE), a menu-based interface to UNIX System V. FACE has been enhanced to be more consistent with a version developed for UNIX System V 386, Release 3.2, and the ease with which applications can be added to FACE has been improved.

For more information about FMLI, see the *Programmer's Guide: Character User Interfaces (FMLI and ETI)*. For information about FACE, see the *User's Guide*.

Graphical User Interface

As part of the effort to encourage a standardization of the UNIX system, Release 4.0 offers a device-independent, portable graphical windowing system, called Graphical User Interface (GUI). GUI is a versatile, user-friendly software interface, composed of several subsystems, called XWIN, X11/NeWS®, and OPEN LOOK™. Each subsystem has a particular function that extends the capabilities of the UNIX operating system.

XWIN

The XWIN Graphical Windowing system is a portable window system that creates a multi-layered server system on top of the UNIX system. XWIN gives the user the ability to create multiple windows on a single display and to run different applications in each window.

XWIN software uses the X protocol for exchanging information between client applications and the graphics server, and Xlib (the C language interface) to built system functions. The X protocol gives application programs running on different systems the ability to communicate with and use or display results from other application windows.

For information about XWIN, see the *Programmer's Guide: XWIN Graphical Windowing System.*

X11/NeWS

X11/NeWS is a second windowing system that runs applications written to the X11 and NeWS protocols. Although the protocols are different, X11/NeWS provides an integrated environment in which both are supported, with both working off a single window manager.

For information about X11/NeWS, see the *Programmer's Guide: X11/NeWS Graphical Windowing System.*

OPEN LOOK

OPEN LOOK™ defines a standard for the appearance and function of the graphical user interface and provides developers with application programmer interface (API) toolkits. API toolkits allow developers to manipulate windows and window-supported graphics to achieve the standard "look and feel" of OPEN LOOK GUI applications. Two toolkits are provided as part of OPEN LOOK—one for writing applications that operate on the XWIN server, and one for writing X11/NeWS applications.

For information about OPEN LOOK, see the *Programmer's Guide: OPEN LOOK™ Graphical User Interface.*

Internationalization

The goal of internationalization is to make it possible for a single program to interact with a variety of users, regardless of the language they speak and the country in which they reside.

Features required for internationalization are

- support for multiple character sets and multi-byte characters
- a message handling facility
- support for different national conventions.

Both ANSI X3.159-1989 and IEEE Std 1003.1 (POSIX) have defined standards for international programs. The ANSI C committee has adopted the term "locale" to refer to a grouping of information that provides behavior dependent on conventions of nationality, culture, and language.

Release 4.0 implements the ANSI and POSIX standards. It provides the hooks necessary for localizing applications, supports multiple locales simultaneously on the same system, and allows multiple instances of the same program to operate each with different locales.

The ANSI C draft standard defines the `setlocale` function, which allows a program to specify the locale to be used for all subsequent locale-specific operations. The `setlocale` function accepts two parameters, an integer indicating the locale-dependent operation that is to be affected (`category`), and the name of the locale (`locale`). Once `setlocale` returns, any of the operations specified in `category` operate according to the designated locale. The variables that can be specified with the `category` parameter are `LC_ALL`, `LC_COLLATE`, `LC_CTYPE`, `LC_MONETARY`, `LC_NUMERIC`, `LC_TIME`, and `LC_MESSAGES`.

(For detailed information about the features described in this section, see the *Programmer's Guide: System Services and Application Packaging Tools.*)

International Character Manipulation

Internationalization corrects some erroneous assumptions held by many users of the ASCII character set—specifically, that a character fits into 7 bits, and that "character" and "byte" are synonymous. In actuality, a single language character may occupy a 7-bit byte (ASCII), an 8-bit byte (European code sets), or a 2- or 3-byte string (Kanji). In Release 3.1, steps were take to remove assumptions from various programs that characters are encoded in 7-bit ASCII. In Release 4.0, UNIX System V offers full support of multiple code sets.

Release 4.0 also supports multi-byte characters—the representation used for international character sets for performing I/O on characters or strings of characters. The multi-byte representation of a single object may (as its name suggests) occupy multiple bytes and include shift state encodings. Multi-byte characters are represented in C language programs as character arrays.

In conformance with the ANSI C standard, Release 4.0 includes support for wide characters—a new data type that can represent every character in any given character set. Wide characters are used in programs to manipulate the *characters* in a file, rather than just the bytes.

Message Management

A difficult problem to address when building international applications is the problem of conversing with the user in his or her native language. To allow the user to input a file name in his or her native language requires the programmer to create new routines that prompt the user with the proper native language phrase and accept the native language as input. Adding support for a language requires finding every locale-dependent area in the code and modifying it to invoke native language statements where appropriate. The time required to build and maintain such programs is enormous, since each supported language requires extensive modifications to the program.

In order to address the problem of locale-dependent messages, programs need to replace references to embedded ASCII strings in program with a call to a general purpose text string look-up service. Release 4.0 provides such a look-up service in two forms: as a C language function and as an executable command, both named `gettxt`.

Given a message identifier, `gettxt` retrieves the text string associated with that identifier. The message database searched by `gettxt` is determined by the current program locale; that is, if the current locale is French, `gettxt` will retrieve the appropriate entry in the French version of the message database.

Once a program has been converted to use `gettxt`, providing support for a new locale is simply a matter of translating the message database for the program into the new language/character set and building the message database with the `mkmsgs` command.

In Release 4.0, the directory `/usr/lib/locale` contains directories for each locale that is supported on a particular system. Each directory for a particular locale contains a subdirectory for messages, named LC_MESSAGES.

Release 4.0 provides two tools, `mkmsgs` and `srchtxt`, to create and search message databases. In addition, Release 4.0 provides the `exstr` tool for converting existing programs to use the message management facilities.

National Conventions

National conventions are the rules and formats we observe when we communicate. Different countries and cultures observe different rules; for example, different countries use different calendars and different formats for communicating the month, day, year, and time of day. For a program to be "international," it must support different calendars and time-of-day calculations, via date/time calculation routines.

In Release 4.0, existing utilities and interfaces have been modified to support both implicit and explicit invocation of different national conventions. Specifically, release 4.0 provides

- a new utility that supports the definition and creation of new collating tables, and two new library routines that use the collating table
- a facility to provide user-definable character classification tables
- generalized date/time editing functions and separate format specifications
- an external variable that can be used by various number conversion routines to edit data in different number formats.

C Language

Most enhancements to the C programming language for this release fall into three categories: conformance with the American National Standards Institute (ANSI) X3.159-1989 C language standard; transition to dynamic shared libraries from static shared libraries; and transition to ELF (Executable and Linking Format) from COFF (Common Object File Format).

Other C language enhancements, resulting from internationalization requirements, are described under the heading "Internationalization" in this chapter.

ANSI C

Three options have been added to the C compiler [see cc(1)] to help make the transition to ANSI C conformance [`-Xt` (transition), `-Xa` (ANSI), and `-Xc` (conformance)]. These options specify the degree of conformance ranging from older compilation systems to the ANSI C standard.

The following topics for migrating from non-ANSI to ANSI C code are covered in the *ANSI C Transition Guide*

- mixing old and new style functions
- functions with varying arguments
- promotions: `unsigned` vs. `value`
- tokenization and preprocessing
- using `const` and `volatile`
- multibyte characters and wide characters
- standard headers and reserved names
- internationalization
- grouping and evaluation in expressions
- incomplete types
- compatible and composite types

Dynamic Linking of C Programs

The C compilation system supports dynamic linking whereby the contents of a shared library are mapped into the virtual address space of processes at run time. External references in the programs are connected with their definitions when the programs are executed. The compilation system provides dynamic linking by default.

As did static shared libraries, dynamic shared libraries save disk storage and system process memory by sharing library code at run time. Unlike static shared libraries, dynamic shared libraries can be fixed or enhanced without having to relink applications that depend on them. Moreover, dynamic shared library code is completely compatible with archive library code. Library builders can use the same source files to create archive and shared object versions of a library. See chapter 2, "C Compilation System" of the *Programmer's Guide: ANSI C and Programming Support Tools* for a discussion of dynamic linking.

In this release, C programs can still be linked with existing static shared libraries, though you should not rely on this feature being in future releases. The `mkshlib` command (used to create a shared library) is no longer supported.

COFF to ELF

A new transparent object file format called ELF (Executable and Linking Format), replaces the old format COFF (Common Object File Format).

COFF files should be converted to ELF files with the `cof2elf` command or by recompiling the source. Recompilation is preferable because it guarantees that executable programs will be compatible with new features in this release. (Note, too, that `cof2elf` discards debugging information.) ELF is described in detail in chapter 13, "Object Files," of the *Programmer's Guide: ANSI C and Programming Support Tools.*

307-321

UNIX
PRESS

A Prentice Hall Title

ISBN 0-13-933821-7