J. Ranade Workstation Series

AIX 6000 DEVELOPER'S TOOL KIT

KEVIN E. LEININGER

CD-ROM ENCLOSED

LEININGER

AIX 6000 DEVELOPER'S TOOL KIT

McGraw Hill

AIX/6000 Developer's Tool Kit

J. Ranade Workstation Series

BAMBARA/ALLEN • *PowerBuilder: A Guide for Developing Client / Server Applications,* 0-07-005413-4

CHAKRAVARTY • *Power RISC System / 6000: Concepts, Facilities, and Architecture,* 0-07-011047-6

CHAKRAVARTY/CANON • *PowerPC: Concepts, Architecture, and Design,* 0-07-011192-8

DEROEST • *AIX for RS / 6000: System and Administration Guide,* 0-07-036439-7

GRAHAM • *Solaris 2.X: Internals and Architecture,* 0-07-911876-3

HENRY/GRAHAM • *Solaris 2.X System Administrator's Guide,* 0-07-029368-6

JOHNSTON • *OS / 2 Connectivity and Networking: A Guide to Communication Manager / 2,* 0-07-032696-7

JOHNSTON • *OS / 2 Productivity Tool Kit,* 0-07-912029-6

LAMB • *MicroFocus Workbench and Toolset Developer's Guide,* 0-07-036123-3

LEININGER • *Solaris Developer's Tool Kit,* 0-07-911851-8

LEININGER • *UNIX Developer's Tool Kit,* 0-07-911646-9

LOCKHART • *OSF DCE: Guide to Developing Distributed Applications,* 0-07-911481-4

PETERSON • *DCE: A Guide to Developing Portable Applications,* 0-07-911801-1

RANADE/ZAMIR • *C++ Primer for C Programmers, Second Edition,* 0-07-051487-9

SANCHEZ/CANTON • *Graphics Programming Solutions,* 0-07-911464-4

SANCHEZ/CANTON • *PC Programmer's Handbook, Second Edition,* 0-07-054948-6

WALKER/SCHWALLER • *CPI-C Programming in C: An Application Developer's Guide to APPC,* 0-07-911733-3

WIGGINS • *The Internet for Everyone: A Guide for Users and Providers,* 0-07-067019-8

AIX/6000 Developer's Tool Kit

Kevin E. Leininger

McGraw-Hill

New York San Francisco Washington, D.C. Auckland Bogotá
Caracas Lisbon London Madrid Mexico City Milan
Montreal New Delhi San Juan Singapore
Sydney Tokyo Toronto

Library of Congress Cataloging-in-Publication Data

Leininger, Kevin E.
AIX/6000 developer's tool kit / Kevin E. Leininger.
p. cm. — (J. Ranade workstation series)
Includes index.
ISBN 0-07-911992-1 (set : hc). — ISBN 0-07-911993-X (set : sc)
1. Operating systems (Computers) 2. AIX (Computer file) 3. IBM RS/6000 Workstation—Programming. I. Title. II. Series.
QA76.76.063L4473 1995
005.26—dc20 95-18964
CIP

The McGraw-Hill Companies

1 2 3 4 5 6 7 8 9 0 DOC/DOC 9 0 0 9 8 7 6 5 (PBK)
1 2 3 4 5 6 7 8 9 0 DOC/DOC 9 0 0 9 8 7 6 5 (HC)

P/N 0-07-037679-4
PART OF
ISBN 0-07-911993-X (PBK)

P/N 0-07-037678-6
PART OF
ISBN 0-07-911992-1 (HC)

The sponsoring editor for this book was Jerry Papke, the editing supervisor was Nancy Young, and the production supervisor was Pamela A. Pelton. This book was set in Century Schoolbook by Carol Woolverton Studio in cooperation with Warren Publishing Services.

Printed and bound by R. R. Donnelley & Sons Company.

Contents

Preface

In my first book, *UNIX Developer's Tool Kit,* I provided a generic introduction to the UNIX development environment and the paradigm of UNIX tools. I also provided a fairly focused overview of the Internet and how you can use it to get software from the Internet. I have included a condensed chapter on the Internet and how to get software from it in this book. However, if you want more information, see *UNIX Developer's Tool Kit* or any of a number of other good books on the Internet.

This book focuses exclusively on AIX tools. The first book provided tools for SunOS and AIX environments as well as a general discussion of other UNIX platforms and their capabilities. This book is exclusively focused on some of the tools available in the AIX environment and how you can get, use, and build them for the AIX environment.

I have added sections on nonnative Internet tools to those in the first book and hope these will be of as much value to you as they have been to me.

Finally, differences between AIX 3.*x* and AIX 4.1 are discussed at some length. There are some significant differences between these two versions, and you should carefully examine them before migrating.

I hope this book assists you with your migration to AIX since it will probably be as arduous and difficult as you are currently thinking. Good luck.

Kevin E. Leininger

Acknowledgments

I would like to thank my wife, Karen, and my sons, Alex and Michael, for their understanding and patience during the production of this book. Only one more to go.

Part

1

AIX: Getting Started

Chapter

1

The AIX Software Development Environment

1.1 The History of AIX

AIX stands for Advanced Interactive Executive and is IBM's primary UNIX offering. Contrary to popular opinion, IBM has been involved with UNIX for many years and, in fact, released the first commercial UNIX offering in the early 1970s. While they never marketed AIX or were successful in selling AIX, it wasn't simply because AIX was inferior. There were obviously other forces at work causing IBM's lack of interest in selling UNIX into their customer base.

AIX was introduced for the workstation platform on the ill-fated RT platform beginning in 1986. The RT was the first Reduced Instruction Set Computer (RISC) IBM offered and came with their new version of System V known as AIX 2.1. For a variety of reasons, the RT was a disaster and IBM sold very few machines. It was underpowered and marketed poorly and to the wrong marketplaces. This failure caused IBM to significantly rethink their UNIX strategy and announce a new machine in 1990 known as the RS/6000.

The initial RS/6000 offerings were targeted at the technical marketplace and subsequently had very good floating-point performance numbers relative to their integer performance. Along with the RS/6000, IBM's first announced AIX version was 3.1 in 1990. They had significant problems with early versions of AIX 3.1, including massive installation problems, standards support issues, and a plethora of kernel problems. In fact, AIX didn't stabilize until Version 3.1.5 in 1992. With this version of AIX, however, IBM had finally hit upon an operating system that was commercially viable and with this version of AIX, the RS/6000 took off.

AIX 2.1 was the first true port of UNIX System V onto the RISC hardware that IBM developed. Underlying the "UNIX-like" component of AIX 2.1 was a lower-level operating system known as Virtual Resource Manager (VRM). The VRM was responsible for managing all the low-level capabilities of the RISC hardware, and the AIX kernel essentially functioned as a guest operating system running on top of VRM (much like a guest operating system can run on top of VM). However, as work begin on AIX 3.*x*, IBM folded much of the low-level VRM technology into the kernel and began to build a low-level, powerful kernel integrated with much more of the higher-level functionality they needed in a commercial operating system.

The initial components of AIX 3.*x* included BSD 4.3, X11, NFS, NCS, and some portions of System V Release 3. Combining all of these different technologies proved to be a significant challenge; however, IBM did a fairly good job in pulling together a commercially viable and stable operating environment.

AIX 3.2.*x* consists of a fairly large kernel which provides rich functionality and increased capabilities such as the virtual memory management subsystem, resource controller subsystems, and other powerful features. It also contained a new, revolutionary filesystem known as the Journalled File System (JFS), which provides a much more reliable and robust filesystem than the standard UNIX filesystem (ufs). There is also an enhanced system administration environment which provides tools such as smit and the Licensed Program Products (lpp) subsystem which make it considerably easier to administer and manage an AIX system. Finally, AIX is very sophisticated as a development environment and includes bundled C compilers (no longer true in AIX 4.1), debugging and performance analysis tools, shared library tools, and sophisticated memory management tools which provide a very sophisticated development environment. All of these capabilities have given AIX a significant competitive advantage with respect to the commercial UNIX environment.

Given the technical market focus of the RS/6000, early machines were sold primarily to engineering and technical markets. However, in 1994 IBM got serious about the commercial capabilities of AIX and started offering machines and software systems that provide a total solution to the commercial marketplace. With the announcement of AIX 4.1, IBM increased their committment to standards and commercial capabilities and now claim to be the leading commercial UNIX vendor in the industry today. Also, their new PowerPC-based technology is offering world-class integer performance for the commercial marketplace. With a new, improved version of AIX and new, extremely powerful processors, IBM is poised to leap forward in the UNIX market.

The other issue which needs to be addressed with AIX 4.1 is the sup-

port for multiprocessing. This will be discussed in more detail later on in the book, but it is important to note that with AIX 4.1, IBM is now providing large symmetric multiprocessing machines to the marketplace. This is important to note for commercial shops that have a need for large multiprocessing solutions. IBM also offers the SP2 machine, which is a large multiprocessing system. It has been largely focused on the technical marketplace with the POWER2 instead of the PowerPC architecture. IBM will continue to focus the SP2 on the technical marketplace but will increase its presence in the commercial marketplace along with their uniprocessor and SMP-based solutions.

1.2 AIX 3.1.5

IBM's first stable version of their operating system was AIX 3.1.5. This was the first release of their operating system that ran reliably and was widely distributed. There are still many machines running AIX 3.1.5 and experiencing no difficulties. However, there were several problems with AIX 3.1.5, and IBM is recommending that users upgrade to at least AIX 3.2.5 or later to begin to take advantage of several new features included in the new version of the operating system. AIX 3.1.5 support from third-party vendors is waning as well, and it is in most users best interest to upgrade as soon as possible.

1.3 AIX 3.2

AIX 3.2.*x* is IBM's most installed UNIX operating system and encompasses most of IBM's recent UNIX-related work. IBM has signficantly enhanced AIX from 3.1.*x* to 3.2.*x,* including increased support for standards, increased kernel sophistication, and significantly enhanced software packaging and support capabilities.

Due to the standards focus of AIX 3.2.*x,* most companies have upgraded from AIX 3.1.5 to AIX 3.2.5. Along with more stable and reliable technology, AIX 3.2.*x* offers a more consistent UNIX implementation which allows for much higher application portability, scalability, and integrity. The standards supported by AIX 3.2.5 are outlined in Sec. 1.5.

AIX 3.2.*x* is clearly the operating system of choice for the near future on the RS/6000 platform due to its reliability and software support. However, if you have higher-performance needs, particularly relating to multiprocessing, or a need for new standards support such as COSE and Single UNIX Specification, you will need to move to the new AIX 4.*x* platform.

There is more discussion of the native development tools and environment of both AIX 3.2.*x* and AIX 4.1 in Chap. 4.

1.4 AIX 4.1

With AIX 4.1, IBM significantly increased its emphasis on the commercial marketplace. With support of COSE and Single UNIX Specification (formerly known as SPEC 1170), symmetric multiprocessing, and other commercial enhancements, IBM is positioning AIX 4.1 as the new standard operating environment for the RS/6000. AIX 4.1 also supports some of the upgraded standards and features which makes it a very powerful UNIX environment. Advanced features of AIX 4.1 include:

NFS 4.2

Motif 1.2

Display Postscript

Logical Volume Manager

Mirroring

Disk Striping

POSIX threads

C2 Security

System V Curses

SOM/DSOM

FDDI

Fibre Channel

ATM

Symmetric Multiprocessing Support

COSE

Common Mode

Single UNIX Specification (formerly known as SPEC 1170)

There are a variety of other standards and features which are being followed by AIX 4.1, and IBM will continue to evolve as the market changes. The standards supported by AIX are described in Sec. 1.5.

The important new standards to software developers in AIX 4.1 are related to Motif, SOM/DSOM, POSIX threads, System V Curses, and the Common Mode.

COSE stands for Common Open Software Environment. COSE was formed by various UNIX vendors in an attempt to create a series of

uniform standards for UNIX. This consists of both operating interface specifications, originally known as SPEC 1170 and a user interface known as Common Desktop Environment (CDE).

AIX 4.1 supports both SPEC 1170 (and Single UNIX Specification soon) and XPG4. XPG4 is based on POSIX 1003.1 and 1003.2 with additions from SVID 3 and other standards. Because some vendors thought that XPG4 was incomplete, SPEC 1170 was born.

Motif 1.2 supports a variety of standards for X-based development which will significantly enhance the portability of X Windows-based applications across platforms. Motif 1.2 supports a variety of new standard X11R5 and beyond constructs which are commonly provided and supported by most, if not all, UNIX vendors.

POSIX 1003.4a Draft 7 threads define a standard way to develop multithreaded applications. These applications can then take advantage of true symmetric multiprocessing machines. These threads also conform to the OSF/1 locking model to ensure portability of the lock code across diverse systems. POSIX thread support ensures that any thread-related code you develop under AIX 4.1 will be supported by most or all of the other UNIX vendors. POSIX thread support is clearly the leader in terms of thread architecture and will be the standard by which all others are judged. AIX 4.1 libp threads implementation is based on the DCE implementation of threads. This will ensure maximum portability across diverse platforms.

AIX 4.1 also provides support for 64-bit-long long int and 128-bit-long double types, which signficantly enhance your ability to manipulate and create portable code.

SOM/DSOM is System Object Model/Distributed System Object Model. IBM is making a hard push into the object-oriented development world with their SOM/DSOM standard. SOM/DSOM allows distributed objects to be shared not only between machines but between different object-oriented languages. Through support of CORBA 2.0 and the future OMG standards, AIX 4.1 will provide a robust object-oriented development platform which ensures portability across different object-oriented environments.

IBM is the first commercial vendor to offer a full implementation of COSE with AIX 4.1. This will provide not only a uniform user interface to the operating system but also a uniform API for developing applications that are GUI related.

System V Curses is the AIX 4.1 update to the libcurses/terminal interfaces package. Because SPEC 1170 was incomplete during AIX 4.1 development, IBM chose System V R31R4 curses because this contained most of the requested changes to the libcurses system.

Finally, Common Mode is an environment which allows binaries to be created that will run across the entire AIX hardware line: PowerPC, POWER2, and POWER. This will allow you to build efficient ex-

ecutables that run on all platforms with a single compilation and link. Common Mode is also available for AIX 3.2.5 and should be utilized immediately to begin to prepare for the newest IBM hardware technology. This is discussed in more detail in Chap. 3.

One important point is that AIX 4.1 maintains binary compatibility with AIX 3.2.*x,* and as such your applications should run virtually unmodified on AIX 4.1. This is critical for the short-term issues involved in the migration to AIX 4.1. There are a variety of changes which occur in AIX 4.1; however, you should be able to continue to operate your applications without a recompile until you deem it necessary to recompile for other reasons.

1.5 Standards

IBM has a clear focus on standards conformance and is working hard on both defining and conforming to industry standards. Standards are continuing to evolve and will mature to some final state sometime in the mid- to late 1990s. The most visible standards are those related to the X/Open group. Single UNIX Specification outlines a set of operating interface calls which will support portability across a variety of platforms. X/Open is also responsible for branding UNIX for all vendors. IBM is clearly the vendor driving the UNIX standardization process.

The other relevant technology is the XPG branding process. The first XPG standard, known as XPG3, has been conformed to by most vendors in the UNIX market. XPG4 is the successor to XPG3 and has become the standard by which all UNIX implementations are judged. Single UNIX Specification was the successor to XPG4 and is more complete than XPG4. As mentioned earlier, however, there are a variety of standards which need to be conformed to in order to be considered UNIX or compatible with UNIX. Contrary to popular opinion, IBM has a very compliant UNIX implementation. In fact, a recent report on standards has IBM in front in most areas. Table 1.1 lists the standards supported. While there are many other standards available for this environment that are not listed in the table, most are implemented with third-party products and are not appropriate for the operating systems environment; therefore, there are standards such as XTI and others which are not discussed.

AIX 4.1 supports all the standards listed in Table 1.1 as well as several new ones, which are listed in Table 1.2. These new standards are the key to ensuring maximum portability for AIX software systems. Through the adherence to standards, AIX 4.1 is providing a platform which will maximize portability and minimize the need for "kludged" code to force portability.

TABLE 1.1 Standards Supported by AIX 3.2.5

Standard	Supported	Standard	Supported
POSIX 1003.1	Yes	PEX 5.1	Yes
POSIX 1003.2	Yes	Motif	Yes
POSIX 1003.4	Yes	X11	Yes
SVID Issue 2	Yes	COSE	Committed to
SVID Issue 3	No	SNMP	Yes
XPG3 Base	Yes	CMIS/CMIP	Committed to
XPG3 Plus	No	ONC/NFS	Yes
XPG4 Base	Committed to	CPI-C	Committed to
OSF/AES	Yes	SQL	Yes
BSD	Yes	DCE	Yes
FIPS 151-1	Yes	SNA-LU6.2	Yes
FIPS 151-2	Yes	CORBA	Yes
SPEC 1170	Committed to	FTAM	Yes
ADA	Yes	X.400	Yes
C	Yes	X.500	Yes
C++	Yes	X.25	Yes
COBOL	Yes	Token Ring	Yes
FORTRAN	Yes	Ethernet	Yes
Pascal	Yes	TCP/IP	Yes
PHIGS	Yes	DECnet	Yes
GKS	Yes	SNA	Yes
GL	Yes	Appletalk	Yes
X11R5	Yes	IPX/SPX	Yes
OpenGL	Yes	LAN Manager	Yes

1.6 AIX Futures

AIX continues to evolve as the market changes. There is a clear focus on standards support and the commercial market. Because of this, technologies such as COSE and others will continue to be supported and will evolve along with their capabilities. A clear move to an object-oriented environment is in the works, and IBM has structured an environment known as PowerOpen. The goal is to provide a microkernel environment with different personalities. For example, OS/2, UNIX, and other operating environments will be supported with a single-kernel environment. This is no different from other companies that are moving to support different environments from within the same operating environment. Finally, the COSE environment will be fully supported to provide commercial UNIX users with the best possible operating environment for their needs.

IBM is one of the leaders in standards support and advanced technology. Clearly, nothing is going to change in the near future with respect to these capabilities or focus issues.

TABLE 1.2 Additional Standards Supported by AIX 4.1

Standard	Supported
XPG4 Base	Yes
COSE	Yes*
Single UNIX Specification (formerly known as SPEC 1170)	Yes*
System V Curses	Yes
FIPS 158-1	Yes
FIPS 160	Yes
FIPS 189	Yes
UNIX93	Yes
POSIX 1003.4a, Draft 7	Yes

*The standard is supported at the current freeze release of the standard. Some of these standards are not yet finalized, and IBM is making every effort to follow and implement the working and final standard.

Chapter

2

AIX Devices

2.1 Introduction

As with any other operating system, it is critically important to understand the devices and how to access and control them on a machine. This is going to be your primary I/O mechanism to the operating system. The fundamental thing to remember about UNIX devices is that they all look exactly the same to the operating system. They are simply bitstreams with no preformatted structure. This makes it easy to write the underlying kernel, but it relies on specific device drivers to access a specific device. Each type of device supports a different type of interface; however, the basic operating is the same. To examine all devices available on the system, use the command:

```
$ lsdev -C
sys0          Available    00-00         System Object
sysplanar0    Available    00-00         CPU Planar
ioplanar0     Available    00-00         I/O Planar
bus0          Available    00-00         Microchannel Bus
sio0          Available    00-00         Standard I/O Planar
scsi0         Available    00-00-0S      Standard SCSI I/O Controller
scsi1         Defined      00-01         SCSI I/O Controller
scsi2         Defined      00-02         SCSI I/O Controller
ent0          Available    00-00-0E      Standard Ethernet Adapter
sysunit0      Available    00-00         System Unit
fpa0          Available    00-00         Floating Point Processor
mem0          Available    00-0C         32 MB Memory Card
fda0          Available    00-00-0D      Standard I/O Diskette Adapter
siokb0        Available    00-00-0K      Keyboard Adapter
siotb0        Available    00-00-0T      Tablet Adapter
sa0           Available    00-00-S1      Standard I/O Serial Port 1
sa1           Available    00-00-S2      Standard I/O Serial Port 2
tok0          Defined      00-03         Token-Ring High-Performance
hdisk0        Available    00-00-0S-00   1.0 GB SCSI Disk Drive
hdisk1        Available    00-00-0S-10   1.0 GB SCSI Disk Drive
lvdd          Available                  N/A
```

```
fd0             Available   00-00-0D-00  Diskette Drive
kbd0            Defined     00-00-0K-00  United States keyboard
hd5             Defined                  Logical volume
hd7             Defined                  Logical volume
tty0            Available   00-00-S1-00  Asynchronous Terminal
hd8             Defined                  Logical volume
hd4             Defined                  Logical volume
hd2             Defined                  Logical volume
hd9var          Defined                  Logical volume
hd3             Defined                  Logical volume
hd1             Defined                  Logical volume
ppa0            Available   00-00-0P     Standard I/O Parallel Port
                                           Adapter
mous0           Defined     00-00-0M-00  3 button mouse
pty0            Available                Asynchronous Pseudo-Terminal
inet0           Available                Internet Network Extension
lo0             Available                Loopback Network Interface
en0             Available                Standard Ethernet Network
                                           Interface
et0             Defined                  IEEE 802.3 Ethernet Network
hdisk2          Defined     00-01-00-00  1.37 GB SCSI Disk Drive
hdisk3          Defined     00-01-00-10  1.37 GB SCSI Disk Drive
hdisk4          Defined     00-02-00-00  1.37 GB SCSI Disk Drive
hdisk5          Defined     00-02-00-10  1.37 GB SCSI Disk Drive
lv00            Defined                  Logical volume
db2space        Defined                  Logical volume
tr1             Stopped                  Token Ring Network Interface
chna0           Defined     00-02        IBM S/370 Channel Emulator/
                                           A Adapter
ppr0            Available   00-03        POWER Gt3i Graphics Adapter
hft0            Available                High Function Terminal
                                           Subsystem
rmt1            Defined     00-00-0S-20  2.3 GB 8mm Tape Drive
cd0             Defined     00-00-0S-50  CD-ROM Drive
rmt2            Available   00-00-0S-60  2.3 GB 8mm Tape Drive
afp0            Defined     00-02-00     IBM S/370 Channel Emulator/
                                           A Printer Driver
rmt3            Defined     00-00-0S-40  Other SCSI Tape Drive
cd1             Available   00-00-0S-50  CD-ROM Drive
rmt0            Defined     00-00-0S-60  5.0 GB 8mm Tape Drive
```

Note that there is a difference between available and defined. Available means that there are actually devices that are attached to the system and are available for use. Defined devices are those contained in the Object Data Manager (ODM) database but are not actually attached to the system. You can only use devices that are available.

As mentioned earlier, all device information is contained and controlled from the system-level database known as the ODM. This contains all predefined and available system-level resources such as devices, communications, and kernel characteristics. It is critically important to understand the basic functioning of the ODM before you can begin to develop in the AIX environment.

There are a series of complex and powerful commands which control the ODM. These are beyond the scope of this book; see the AIX InfoExplorer system for more details on the ODM.

You can query a specific type of device with the command:

```
$ lsdev -C -c tape
rmt1          Defined       00-00-0S-20   2.3 GB 8mm Tape Drive
rmt2          Available     00-00-0S-60   2.3 GB 8mm Tape Drive
rmt3          Defined       00-00-0S-40   Other SCSI Tape Drive
rmt0          Defined       00-00-0S-60   5.0 GB 8mm Tape Drive
```

This tells you that there are four defined tape devices, rmt0 through rmt3; however, only one is actually attached to the system and available: rmt2. This means that you need to access the tape drive with the filename (a.k.a. device name) /dev/rmt2. The section below outlines this in more detail.

2.2 Tape Devices

Tape devices are your primary external I/O mechanism to the operating system. Each UNIX implementation has a different convention for naming the device interfaces, and AIX is no different. As mentioned above, you need to query the ODM database to find out which tape device is actually available and to discover the associated name. Once you have found the available tape device name, there are some specifics you must be aware of. The basic syntax for the complete device name is:

```
/dev/rmtn.m
```

where n is the logical device name assigned when the kernel is built.
m controls the behavior and characteristics of the device.

Again, to examine the tape devices on the system, use the command:

```
$ lsdev -C -c tape
rmt1          Defined       00-00-0S-20   2.3 GB 8mm Tape Drive
rmt2          Available     00-00-0S-60   2.3 GB 8mm Tape Drive
rmt3          Defined       00-00-0S-40   Other SCSI Tape Drive
rmt0          Defined       00-00-0S-60   5.0 GB 8mm Tape Drive
```

This shows that the tape device available is /dev/rmt2, and you can access it using this filename. To control specific types of behavior on the tape device, you must use the full filename as specified above. The suffix on the device name (designated by the m in the syntax description above) controls the behavior of the device itself. There are eight possible suffix values as shown in Table 2.1. Each category controls three particular aspects of the device: (1) rewind on close specifies that the tape is rewound after each command operation, (2) retension on open specifies that the tape should be retentioned before each command op-

TABLE 2.1 Tape Special File Characteristics

Filename	Rewind on close	Retension on open	Bytes per inch
/dev/rmt*	Yes	No	Density setting #1
/dev/rmt*.1	No	No	Density setting #1
/dev/rmt*.2	Yes	Yes	Density setting #1
/dev/rmt*.3	No	Yes	Density setting #1
/dev/rmt*.4	Yes	No	Density setting #2
/dev/rmt*.5	No	No	Density setting #2
/dev/rmt*.6	Yes	Yes	Density setting #2
/dev/rmt*.7	No	Yes	Density setting #2

eration, and (3) bytes per inch specifies a density based on the type of tape device physically attached to the processor. Each of these can be used to control a particular tape device and the type of behavior you wish. For example, to use a device and not rewind the tape after the operation, use the command:

```
$ mt -f /dev/rmt0.1 fsf 1
```

This will skip the first file on the tape and leave the tape positioned at the end of the first file. Note that the default device (with no suffix) will rewind the tape at the end of each operation. This means that if you execute the following command:

```
$ mt -f /dev/rmt0 fsf 1
```

you will skip the first file, the mt command will end, and the operating system will rewind the device to the beginning of the tape. Obviously, this defeats the purpose of the skip command in the first place and needs to be clearly understood.

If you have two tar files on tape and you need to access both, you need to use a series of commands such as:

```
$ tar xvf /dev/rmt0.1
$ tar xvf /dev/rmt0
```

This will read the first tar file, pause the tape, and then read the second tar file on the tape. Again, if you simply use the device name /dev/rmt0, you would simply read the first tar file twice. Keep this in mind as you begin to use tape devices.

For more information on the rmt devices, see the man page on rmt.

2.3 CD Devices

AIX has been behind with respect to CD support. The UNIX standard has been High Sierra with support for long filenames and other non-ISO constructs. AIX 3.2.5 does not fully support the High Sierra standard, but AIX 4.1 does. This may be one reason to upgrade to AIX 4.1.

To see what CD devices are defined and available on your machine, use the command:

```
$ lsdev -C -c cdrom
cd0             Defined       00-00-0S-50   CD-ROM Drive
cd1             Available     00-00-0S-50   CD-ROM Drive
```

This tells you that /dev/cd1 is available and that this is the filename you should use to access the CD device.

Given its lack of High Sierra support, you may have to play some games with filenames and extensions to get most of your CDs to work correctly. The command you use to mount up a CD is:

```
$ mount -r -v cdrfs /dev/cd1 /cdrom
```

where /cdrom is an arbitrary mount point, and /cdrom is an empty directory which you can create anywhere.

2.4 Disk Devices

The disk structure in AIX is different than in most other versions of UNIX. AIX has structures called logical volumes which consist of logically defined partitions which may or may not correlate to physical disks. Logical volumes can span physical volumes or be wholely contained within one physical disk. They can also be mirrored and striped to enhance reliability and performance. This entire filesystem structure is called the JFS.

While it is beyond the scope of this book to document the AIX JFS, there are a few basic concepts you need to understand in order to help your system administrator manage your disk space effectively.

To see what disk drives are available on your system, you can use the command:

```
$ lsdev -C -c disk
hdisk0          Available     00-00-0S-00 1.0  GB SCSI Disk Drive
hdisk1          Available     00-00-0S-10 1.0  GB SCSI Disk Drive
hdisk2          Defined       00-01-00-00 1.37 GB SCSI Disk Drive
hdisk3          Defined       00-01-00-10 1.37 GB SCSI Disk Drive
```

As you can see, this machine has four disk drives defined but only two attached. This means that there are probably external disks that have

been removed for any number of reasons. Once you have examined the physical disks attached to the system, you need to understand the logical volume structure of the disks before you can make intelligent decisions concerning software placement. Use the command:

```
$ lsvg
dbasevg
rootvg
```

This shows that there are two volume groups on this system. Each of these volume groups consists of one or more logical volumes. To get information about a particular volume group use the command:

```
$ lsvg rootvg
VOLUME GROUP:    rootvg       VG IDENTIFIER:   00000044e0f15b11
VG STATE:        active       PP SIZE:         4 megabyte(s)
VG PERMISSION:   read/write   TOTAL PPs:       496 (1984 megabytes)
MAX LVs:         256          FREE PPs:        202 (808 megabytes)
LVs:             11           USED PPs:        294 (1176 megabytes)
OPEN LVs:        10           QUORUM:          2
TOTAL PVs:       2            VG DESCRIPTORS:  3
STALE PVs:       0            STALE PPs        0
ACTIVE PVs:      2            AUTO ON:         yes
```

This tells you that the rootvg volume group is active, has physical partition sizes of 4MB, is read/write, and consists of 496 4MB physical partitions, giving you a total capacity of 1.984GB worth of storage.

rootvg is a special filesystem (volume group) which exists on all AIX machines and which has special tools for backup and recovery. It is usually a good idea to create a different volume group to contain special software systems such as databases and other applications. For example, the other volume group shown above is dbasevg. This volume group contains a database system which is separate from the rootvg volume group. This has a variety of advantages, including separate backup and recover, maintainability, and reliability. You can vary the volume group and other volume specific activities on- and off-line.

The bottom line is that it is a good idea to create separate volume groups for your software systems. Note that it is a maintenance headache if you create too many volume groups, and you should attempt to cluster as many software systems together as possible to decrease the maintenance load on the system administrator.

This is certainly not an exhaustive discussion of the disk subsystems on AIX. It does, however, provide you with enough information to discuss the disk allocation and usage issue with your system administrator intelligently and come to some mutual agreements on an overall storage strategy.

Chapter

3

Program Development Under AIX

3.1 Introduction

AIX is a very sophisticated development environment that includes a variety of features and functions not found in other development environments, even other versions of UNIX. This chapter will outline some of the basic capabilities and features of the AIX development environment. It is by no means an exhaustive presentation of the capabilities of AIX but merely represents the most basic and fundamental AIX offerings.

It is important to note that AIX 3.2*x* is the topic of the general sections in this chapter. AIX 4.1 information is covered in specific sections that are noted as 4.1 sections. AIX 4.1 is compatible with AIX 3.2.*x* unless otherwise noted.

3.2 Linking and Loading

One of the advantages of the AIX operating system is its ability to link and load dynamically. Unlike many other operating systems, AIX can dynamically include pieces of the executable at run-time instead of at compile- and link-time. This provides several advantages:

Smaller executable size

Less memory usage

Ease of program maintenance

More dynamic kernel and program capabilities

The logical counterpart to dynamic linking is static linking. Statically linked applications simply include all executable code from all referenced objects into a single executable file. This results in large executables which are resource intensive and relatively inflexible. With static linking you have lost the ability to dynamically change large portions of your executable without so much as a recompile. There are reasons to use static linking, however, and people who develop real-time systems and other performance-intensive systems can tell you what they are. However, it is a rare situation where a statically linked system is a better choice than a dynamically linked one.

The basic capabilities of this dynamic development environment extend to the AIX kernel itself and provide significant advantages over most other operating systems. AIX, due to its nature as a dynamically built kernel, provides:

Dynamic kernel extensions

System call modification and extensions

Dynamic device driver configurations

These make AIX a very flexible and powerful development environment.

The ability to dynamically load and link an executable means that you can created shared libraries which can be changed with little impact to the production system. This means that you can create a library which can be simultaneously shared by more than one running application. This is key to aiding in software developer productivity and code quality.

At this point it is appropriate and necessary to talk about the compiler technology on AIX, known as the XL compiler technology. (Note that this has changed in AIX 4.1, see Section 3.2.1 for specific information.) All of IBM's compilers generate an exact intermediate language from their compilers known as XIL. From this, the specific compiler generates the appropriate codes for the individual platform on which it was generated. The basic steps in the compilation process are:

1. *Language analysis.* This tokens (breaks apart) all parts of the language syntax in order for the optimization and internalization to occur. This step creates the XIL intermediate codes that are similar across different IBM platforms.
2. *Optimization.* The optimization phase manipulates the previously created XIL codes to enhance performance.
3. *Register allocation.* This phase of the compilation creates an intermediate format based on a limited number of registers on a given machine and begins to form the executable to the specific machine.

4. *Assembly to object format.* This is the final step in the process where the compiler generates machine-specific codes which are then linked into an XCOFF format file.

These steps are common across all IBM XL compiler technologies. Given this similarity across platforms, IBM can generate more flexible, portable systems in any language. XIL and the common intermediate executable architecture are why it is easy to link C, Fortran, Pascal, and other languages together on an AIX machine.

Under AIX, creating a dynamically loaded application is very simple. By using specific options on the compilation statement, you can create both dynamic libraries as well as applications that use them. A simple example including code segments is shown below:

```
/* This is prog.c */
#include <stdio.h>
int common;
main() {
       printf("This is a shared library function example\n");
       common();
       printf("This example is complete\n");
       }
```

The above segment outlines a file called prog.c, which is the C source file for the main program we are going to build. Next you need to create a file known as an import file (let's call this file common.imp). The following is an example of such a file:

```
(common.imp)
#! ./commonobj
common
.
.
.
```

The first line defines the name of the import file. The second line defines the name of the object file in which the succeeding shared objects will be found; in this case the object filename is commonobj. The third and following lines define the shared objects contained in the file commonbj (in this case only common).

The basic structure of an import file is as follows:

1. Object path
2. Object filename
3. Object members
4. Shared symbols

The syntax looks like:

```
*This is a comment
*
#! path/file (member)
symbol1
symbol2
...
symboln
```

where *—comments always begin with an * and are ignored.
#! defines the first line containing the definition of the import path.
path/file is a fully qualified or relative path and filename to the shared object.
member specifies a shared object within a shared library (if one exists).
symbol1 to symboln defines all shared symbols within the shared object.

The basic syntax is the same for all import files and needs to be followed explicitly to ensure the correct results.

To compile the prog.c application and link in the shared objects, use the command:

```
$ cc -b import:common.imp -oprog prog.c
```

This will create an executable file named prog which has an external object common() that will be located in a file named commonobj.

Next we need to build the file which contains the shared code known as commonobj. The source code for a program known as common.c is shown below:

```
#include <stdio.h>
/* This is common.c */
int common();
{
       printf("We are inside the shared function known as common\n");
}
```

You compile this with the command:

```
$ cc -c common.c -o common.o
```

You now need to generate a file similar to common.imp but one which tells the linker which files are being exported from the file common.c. This file is named common.exp, and it contains the following code:

```
#! ./commonobj
common
```

Again, the first line defines the name of the shared object file, while the subsequent line defines the objects to be shared from the file commonobj. The structure of the export file is identical to that of the import file as described above and all issues are the same regarding the strict adherence to the syntax and structure of the file. Now you link the final shared object file with the command:

```
$ ld -H512 -T512 -bglink:/lib/glink.o -b export:common.exp -bM:SRE
-ocommonobj -lc common.o
```

The ld command invokes the loader portion of the XL compiler system on AIX. It servers only as a linker and not as a compiler. Note that this step doesn't require any code to be compiled but merely linked into a form which other object files can be linked against. The important options in the command above are described as follows:

`-H512`	An option which defines text, data, and loader sections of the output file. This number is the boundary number of the output file.
`-T512`	512 marks the starting address of the text segment in the resulting object file.
`-bglink:/lib/glink.o`	/lib/glink.o is a prototype file used by the linker to generate code for all defined external references.
`-b export:common.exp`	This references the export file as described above.
`-bM:SRE`	This defines the object output type. There are other options for this; see the manual page for more details.
`-ocommonobj`	This outputs to a file named commonobj.
`-lc`	This is used to link the library /lib/libc into the executable.
`common.o`	This is the object file to load.

Once you have created the shared library as above, you can execute your main program as created in the initial compile:

```
$ ./prog
```

This will invoke the executable prog that is dynamically linked to the commonobj object. Thus, when prog calls common during execution, the loader will dynamically load and execute the code from commonobj. Note that the loader determines if commonobj has been loaded into memory previously. If it has, it uses the current in-memory copy; if not, it loads and executes the copy from disk.

You can place shared objects into a common respository known as a library by simply using the UNIX archive facility. A simple example is:

```
$ ar q commonlib .a commonobj.o
```

The syntax to include a shared object from a library is very similar to that used to include a shared object. The exp file as described earlier might look like:

```
#! commonlib.a(commonobj.o)
common
```

and in the input file you might use might look something like:

```
(common.inp)
#! commonlib.a(commonobj.o)
common
```

Finally, to link and use an object from a shared library, you would use a command like:

```
$ cc -oprog prog.c
$ ld -b export:common.imp -o prog -lc prog.o
```

This will create an executeable named prog that is dynamically linked to an object named common in the shared library commonlib.a.

This section has presented the basic commands necessary to create and use the dynamic capabilities of the XL compiler subsystem on AIX. There are many other capabilities and characteristics under AIX which are more fully described in your system documentation.

3.2.1 AIX 4.1 specific linking and loading information

The first and most important thing to note about AIX 4.1 is that the XL C compiler is no longer included with AIX. You now must buy a separate product know as C for AIX. XL C will not run under AIX 4.1. This places a larger emphasis on using either gcc or some other equivalent tool unless you have explicitly ordered C for AIX with your AIX 4.1 release. Be careful of this!

The major differences between XL C and C for AIX are as follows:

1. C for AIX implements much tighter ANSI conformance including:
   a. You can no longer mix K&R and ANSI function prototypes.
   b. You can no longer create null dimension multidimensional arrays
   c. Tags introduced at the parameter scope are not exported to the enclosing nonparameter scope.
2. Parameter evaluation order has changed; this is not portable across various compilers.
3. Preprocessor differences including:
   a. Output preserves coordinate of each token.

   b. No redundant #LINE directives or multiple successive blank lines in output.
   c. Erroneous or incomplete macro invocations are expanded.
4. C for AIX requires that #pragma options align=opt appear before the structure definition, while XL C allowed it anywhere before the closing brace of the structure definition.
5. C for AIX supports a long long type.
6. C for AIX does not define _ANSI_C_SOURCE by default.
7. References to array out of bounds information react differently. C for AIX does not pad arrays like XL C did.
8. Uninitialized variables may be different between XL C and C for AIX.
9. C for AIX behaves differently with pointer arithmetic. Differences due to optimization may occur during pointer arithmetic, so be careful.

Those are the major differences in terms of language support. There are other differences in compiler modes, linking capabilities and other issues as outlined in later sections.

3.3 Process-Related System Calls

AIX fully supports most accepted UNIX standards for system interface calls and definitions. However, since the standards are still being defined, IBM was forced to make some decisions regarding interface support and definitions. This section outlines some of the major differences between AIX system calls and other UNIX system calls. It merely represents the major differences and certainly doesn't present all relevant system call information. Consult your system documentation for more information regarding the hundreds of other system calls available to you under AIX.

UNIX is based on the concept of the fork, the system call which creates a new process. By forking new processes, UNIX creates an environment which isolates individual programs and processes to avoid conflict. The basic system call is the fork() system call. This will create an entirely new process with its own process context and content.

Along with the fork system call, UNIX uses the exec() system call. This copies executable information from disk to memory. Unfortunately, the exec() process writes any disk data into your current process context, overwriting your current process-level information. This is obviously unacceptible since it will erase everything you are doing in your current process. That is why exec() is normally used with fork() to first

create an entirely new process, then load some executable information into the newly created process, and execute from there. When that process is finished, you are returned to your current process context for continuing execution.

The process of exec() is extremely resource intensive and often wasteful in its use of system capabilities. IBM realized this and has created several system calls (see Table 3.1) which provide an alternative to this paradigm.

The load() system call loads an individual object file into a known shared space in your current process context. This makes this object available to your current process without requiring the overhead of forking a new process and "exec-ing" an entirely new executable into memory. There are many additional things which must be loaded into memory for an entirely new process that are not required by a simple executable "load" operation. This provides a significant advantage to AIX over many other versions of UNIX and many other operating systems.

The unload() system call unloads an individual object from the shared section of a process' memory. This is useful for memory cleanup. An object can only be unloaded when it is not in use or has another object importing symbols from it.

The loadbind() system call is used if you have set the L_NOAUTO-

TABLE 3.1 AIX Specific System Calls

Call	Definition	Function
load	#include <sys/ldr.h> int (*load(ObjectFile,LoadFlags,LibraryPath))() char *ObjectFile; unsigned int LoadFlags; char *LibraryPath;	Provides individual object file loading capabilities
unload	#include <sys/ldr.h> int unload (ObjectPointer) int (*ObjectPointer)();	Unloads an individual object file from memory
loadbind	#include <sys/ldr.h> int loadbind(BindFlag,ExportPointer,ImportPointer) int BindFlag; void *ExportPointer; void *ImportPointer;	Resolves all outstanding references between loaded objects
loadquery	#include <sys/ldr.h> int loadquery(QueryFlags,DataBuffer,BufferLen); int QueryFlags; void *DataBuffer; void *Bufferlen;	Provides information regarding errors on system calls as well as objects in memory

DEFER in the load system call routines. If you choose the resolve references upon object load, you may not need loadbind(). However, you will need to use loadbind() to resolve any unresolved references if all calls to load include the L_NOAUTODEFER flag in their execution. This means that the loadbind() system call will resolve any unresolved references in the executable upon execution of the loadbind() system call itself. See the manual pages for more on the capabilities of loadbind().

Finally, the loadquery() system call will provide information on currently loaded objects as well as errors from the other related system calls such as load(), unload(), loadbind(), and exec().

3.4 Memory Management System Calls

AIX has a very unique memory management subsystem which has caused significant debate among UNIX developers. Suffice it to say that AIX has some very sophisticated memory management capabilities which should be understood and exploited carefully, if at all. However, since most AIX developers will need to understand some of the basic capabilties of AIX, the basic ones are mentioned in this section. While the subsystem itself is unique, the memory management calls to the operating system itself are not. The standard ones supported by AIX are shown in Table 3.2 broken into several functional areas.

While dynamic memory allocation is recommended, there are other AIX calls which can be used to manipulate memory and the process context on which you are operating. Two such system calls are brk() and sbrk(). These calls are used by the above calls to obtain additional memory from the kernel allocator/deallocator. It is this process which is responsible for the actual distribution and management of dynamic memory space. It is strongly recommended that you use the system calls in Table 3.2 instead of brk() and sbrk() to ensure the proper operating of your memory allocation process.

AIX has a particular memory management algorithm which seems to give software developers unexpected results. When the AIX virtual memory system crosses a threshold known as a highwater mark, the kernel begins to send messages to processes with the highest memory utilization current in memory. It first send a warning signal on which it expects the program to act. If the memory situation worsens, the kernel will send a KILL signal to a high-memory-usage process. This means that this process is KILLed (SIG 9) without any recovery. This can be disasterous to a large memory program. It is important to code your programs to check for signals from the kernel regarding memory. See the AIX system documentation for more information on how to do this.

TABLE 3.2 AIX Specific System Calls

Call	Definition	Function
malloc	#include <sys/types.h> #include <malloc.h> void *malloc(MemSize) size_t MemSize;	Allocates a specific piece of memory
alloca	#include <sys/types.h> #include <malloc.h> void *alloca(MemSize) int MemSize;	Allocates a specific piece of memory and frees it automatically when finished using it
calloc	#include <sys/types.h> #include <malloc.h> void *calloc(NoOfElements,ElementSize) size_t NoOfElements size_t ElementSize	Allocates space for the number of array elements specified by NoOfElements
valloc	#include <sys.types.h> #include <malloc.h> void *valloc(MemSize) unsigned int MemSize;	BSD equivalent to malloc but doesn't return a pointer to the first correctly aligned address but instead to the first data character (this can cause problems; use malloc)
free	#include <sys/types.h> #include <malloc.h> void free (MemoryPointer) void *MemoryPointer;	Frees up memory reserved by on of the allocations calls listed above
realloc	#include <sys/types.h> #include <malloc.h> void *realloc(MemPointer,NewSize) void *MemPointer; size_t NewSize;	Reallocates current memory segment to NewSize from the current size

3.5 The XL C Compiler

One of the advantages of AIX3.2.*x* is the inclusion of the XL C compiler with native AIX 3.2.*x*.

IBM is known for its best of breed compiler technology, not the least of which is their XL series of compilers. The XL compiler system paradigm was discussed earlier in this chapter and will not be discussed again. However, it is fair to remind you that the XL compiler system generates highly efficient code optimized for a particular piece of hardware while still maintaining compatibility across a variety of hardware platforms and third-generation languages.

The XL C compiler is include with AIX since most of AIX is written in C. In fact, UNIX and C are ubiquitous in that most of UNIX is written

in C to ensure portability and performance. Because of this, the XL C compiler is the most important native compiler technology available with AIX. This section outlines its basic capabilities as well as characteristics to give you a better understanding of the native AIX development environment.

As with most other lpp, there are environment variables which must be set properly to use the XL C compiler system. The first is the LANG variable, which defines the language in which the compiler messages are to be presented. LANG should be set to En_US in the United States. Check your local documentation for details outside the United States. The other variable to be set is NLSPATH. This should be set to /usr/lpp/msg/%L/%N. These variables have probably been defined by your system administrator. Check before setting them explicitly in your .profile or .cshrc files.

The XL C compiler system uses a default configuration file /etc/xlc.cfg. The actual program executed when you invoke the XL C compiler is /usr/lpp/xlc/bin/xlcentry. When xlcentry is invoked, it references the /etc/xlc.cfg file, which defines the default options for both the compiler and linkage editor based on things called stanzas.

There are three default symbolic links to the same executable for the XL C compiler system: /bin/cc, /bin/xlc, and /bin/c89. Each of these references a stanza in the /etc/xlc.cfg file. The default /etc/xlc.cfg looks like:

```
* @(#) xlc.cfg 1.12 9/16/93 04:05:19
*
* COMPONENT_NAME: (CC) AIX XL C Compiler
*
* FUNCTIONS: C Configuration file
*
* ORIGINS: 27
*
* (c) COPYRIGHT IBM CORP. 1989,1993
* All Rights Reserved
* Licensed Materials - Property of IBM
*
* US Government Users Restricted Rights - Use, duplication or
* disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
*
* ansi c compiler
xlc:     use           = DEFLT
         crt           = /lib/crt0.o
         mcrt          = /lib/mcrt0.o
         gcrt          = /lib/gcrt0.o
         libraries     = -lc
         proflibs      = -L/lib/profiled,-L/usr/lib/profiled
         options       = -H512,-T512,-D_ANSI_C_SOURCE,-qansialias

* extended c compiler aliased as cc
cc:      use           = DEFLT
         crt           = /lib/crt0.o
         mcrt          = /lib/mcrt0.o
```

```
        gcrt          = /lib/gcrt0.o
        libraries     = -lc
        proflibs      = -L/lib/profiled,-L/usr/lib/profiled
        options       = -H512,-T512,-qlanglvl=extended,-qnoro,
                        -qnoroconst

* ansi c compiler aliased as c89
c89:    use           = DEFLT
        crt           = /lib/crt0.o
        mcrt          = /lib/mcrt0.o
        gcrt          = /lib/gcrt0.o
        libraries     = -lc
        proflibs      = -L/lib/profiled,-L/usr/lib/profiled
        options       = -H512,-T512,-D_ANSI_C_SOURCE,-qansialias

* ansi c compiler aliased as xlc_r
xlc_r:  use           = DEFLT
        crt           = /lib/crt0_r.o
        mcrt          = /lib/mcrt0.o
        gcrt          = /lib/gcrt0.o
        libraries     = -L/usr/lib/dce,-lc_r,-lpthreads
        proflibs      = -L/lib/profiled,-L/usr/lib/profiled
        options       = -H512,-T512,-D_ANSI_C_SOURCE,-qansialias,
                        -D_THREAD_SAFE,-D_CMA_NOWRAPPERS_

* extended c compiler aliased as cc_r
cc_r:   use           = DEFLT
        crt           = /lib/crt0_r.o
        mcrt          = /lib/mcrt0.o
        gcrt          = /lib/gcrt0.o
        libraries     = -L/usr/lib/dce,-lc_r,-lpthreads
        proflibs      = -L/lib/profiled,-L/usr/lib/profiled
        options       = -H512,-T512,-qlanglvl=extended,-qnoro,
                        -D_THREAD_SAFE,-D_CMA_NOWRAPPERS_,-qnoroconst

* common definitions
DEFLT:  xlc           = /usr/lpp/xlc/bin/xlcentry
        as            = /bin/as
        ld            = /bin/ld
        options       = -D_IBMR2,-D_AIX,-D_AIX32,-bhalt:4
        ldopt         = "b:o:e:u:R:H:Y:Z:L:T:A:V:k:j:"
```

Note that each section defines a different usage of the XL C compiler system. The basic options for each are the same, however, and are outlined below:

`as`	Defines the pathname for the assembler.
`asopt`	A list of options on the command line directed to the assembler and not the compiler.
`crt`	The pathname of the C run-time object passed as the first parameter to the linker (also known as the linkage editor).
`csuffix`	The default suffix used for C source programs.
`gcrt`	An alternative C run-time object used if -pg is on the command line.
`ld`	Defines pathname for the linker.
`ldopt`	Defines the options passed to the linker.
`libraries`	Specifies the comma-separated flags passed from the XL C compiler to the linker.

libraries2	Is similar to libraries, but each parameter specified must have a profiled version in the stanza proflibs.
mcrt	An alternative C run-time object used if the -p option is on the command line.
options	Command line options.
osuffix	Default suffix for object files.
proflibs	Comma-separated parameters which specify profiled versions of the libraries specified in the stanza libraries2.
ssuffix	The default suffix for assembler files.
use	Use option values are specified after the stanza name; any comma-separated options are added to those in the specified stanza, and options without commas apply only if no value for that stanza is specified locally in the current stanza.
xlc	The pathname to the XL C compiler.
xlcopt	The options are specified to the XL C compiler only when seen on the command line.

Each of the above options can be specified for a particular invocation of the XL C compiler. In other words, you can create your own symbolic link to /usr/lpp/xlc/bin/xlcentry and define a stanza in /etc/xlc.cfg to control the compiler options, environment, and interface to the other systems such as the assembler and linker.

Based on the invocation of the XL C compiler, you are requesting varying levels of the C standard. You can also specify a particular language level by using the -qlanglvl option on the command line. For example:

```
$ cc -qlanglvl=ansi prog.c
```

This specifies that you should compile prog.c using the ANSI standard conventions and not some other level, such as SAA. There are four basic C language levels supported:

ansi	ANSI standard
saa	SAA current standard
saal2	SAA Level 2 standard
extended	Older Kernighan and Ritchie (K&R) style

The basic syntax for the XL C compiler system is as follows:

```
cc [ option | file ]...
xlc [ option | file ]...
c89 [ option | file ]...
```

The cc, xlc, and c89 commands compile XL C source files. These commands are the same except for the default language level. For cc, the default language level is extended. For xlc and c89, the default language level is ansi. These commands also process assembler source

files and object files. Unless the -c option is specified, these commands call the linkage editor to produce a single object file. Input files can be any of the following:

1. Filename with .c suffix: C source file
2. Filename with .i suffix: preprocessed C source file
3. Filename with .o suffix: object file for ld command
4. Filename with .s suffix: assembler source file

There can be one or more options. Flag options are:

-#	Displays verbose information on the compiler's progress without invoking anything.
-B<prefix>	Constructs alternate compiler/assembler/linkage editor program names. <prefix> is added to the beginning of the standard program names.
-c	Compiles only; does not call ld.
-C	Writes comments to output when doing preprocessing; used with -E and -P.
-D<name>[=<def>]	Defines <name> as in #define directive. If <def> is not specified, 1 is assumed.
-E	Preprocesses but does not compile; outputs to stdout.
-F<x>[:<stanza>]	Uses alternate configuration file <x> with optional <stanza>. If <stanza> is not specified, xlc is assumed.
-g	Produces debug information.
-I<dir>	Searches in directory <dir> for include files that do not start with an absolute path.
-l<key>	Searches the specified library file, where <key> selects the file lib<key>.a.
-L<dir>	Searches in directory <dir> for files specified by -l<key>.
-M	Creates an output file suitable for inclusion in a description file for the UNIX make command.
-o<name>	When used with -c, names the .o fil<name>; otherwise names the executable file <name> instead of a.out.
-O	Optimizes generated code.
-O2	Equivalent level of optimization as -O in the previous release.
-O3	Performs some memory- and compile-time-intensive optimizations in addition to those executed with -O2. The -O3-specific optimizations have the potential to alter the semantics of a user's program. The compiler guards against these optimizations at -O2, and the option -qstrict is provided at -O3 to turn off these aggressive optimizations.
-p	Generates simple profiling support code.
-pg	Generates profiling support code. Provides more extensive profiling than -p.
-P	Preprocesses but do not compile; outputs to .i file.
-Q<x>	In-lines all appropriate functions where x can be one of the following:

	!—Does not in-line any function =<lc>—In-lines if number of source statement in function is less than the number specified in <lc> -<nm>—Does not in-line function listed by names in <nm> +<nm>—Attempts to in-line function listed by names in <nm>
-S	Produces a .s file for any source file processed by the compiler.
-t<x>	Applies prefix from the -B option to the specified program <x>, where x can be one or more of the following: p = preprocessor, c = compiler, a = assembler, and l = linkage editor.
-U<name>	Undefines name as in #undef directive.
-v	Displays verbose information on the compiler's progress.
-w	Suppresses informational, language-level, and warning messages.
-y<x>	Specifies compile-time rounding of constant floating-point expressions, where <x> can be one of the following: n = round to nearest, m = round toward minus infinity, p = round toward positive infinity, and z = round toward zero.

Other options are specified as follows:

```
-q<option>
```

where <option> is an on/off switch such that, if *x* is the option, -q*x* turns the option on, and -qno*x* turns the option off. For example, -qsource tells the compiler to produce a source listing, and -qnosource tells the compiler not to produce a source listing.
The following override the initial compiler settings:

ansialias	Specifies type-based aliasing to be used during optimization.
attr	Produces attribute listing (only referenced names).
compact	Reduces code size where possible, at the expense of execution speed. Code size is reduced by inhibiting optimizations that replicate or expand code in-line.
cpluscmt	Permits // to introduce a comment that lasts until the end of the current source line, as in C++.
dbxextra	Generates symbol table information for unreferenced variables. By default such information is not generated, thus reducing the size of the executable compiled with the -g option.
extchk	Performs external name type-checking and function.
idirfirst	Specifies the search order for files included with the #include file_name directive. Uses -qidirfist with the -Idirectory option. If -qidirfirst option is specified, the directories specified by the -Idirectory option are searched before the directory in which the current file resides.
inlglue	Generates fast external linkage by in-lining the code (pointer glue code) necessary at calls via a function pointer and calls to external procedures.
list	Produces object listing.
listopt	Prints settings of all options in listing.
mbcs	String literal and comments can contain MBCS characters.
noprint	Directs listing to /dev/null.

`nostdinc`	Specifies which files are included with the #include "file_name" and #include <file_name> directives. If -qnostdinc is specified, the /usr/include directory is not searched.
`ro`	Does not put string literals in read-only area.
`phsinfo`	Displays phase information on the screen.
`proto`	Asserts that procedure call points agree with their declarations even if the procedure has not been prototyped. This allows the caller to pass floating-point arguments in floating-point registers instead of general-purpose registers.
`ro`	Places string literals in read-only area. This option is the default for xlc.
`roconst`	Places static external and global identifiers that are const qualified in the read-only area.
`source`	Produces source listing.
`srcmsg`	Specifies that source lines are to be displayed with the message, with pointers to the column position of the error.
`statsym`	Adds user-defined nonexternal names that have static storage class to the name list (the symbol table of xcoff objects).
`strict`	Valid only at -O3. This option turns off aggresive optimizations which have the potential to alter the semantics of a user's program. This option also sets -qfloat=nofltint: norsqrt.
`tocdata`	Places scalar external data of one word or less in the TOC (table of contents). If this option is not on, the address of scalar external data is placed in the TOC. This requires an extra load of the address before accessing the data.
`xcall`	Generates code to static routines within a compilation unit as if they were external routines.
`xref`	Produces cross-reference listing (only referenced names). For example:

`-q<option>=<suboption>`
`-q<option>=<suboption1>:<suboption2>:...:<suboptionN>`

where <option> is assigned a specific suboption value or list of suboption values as follows:

`align=<algnopt>`	Specifies one of the following three alignment rules: (1) POWER architecture (align=power, default), (2) 2-byte alignment (align=twobyte), or (3) 1-byte alignment (align=packed) for aligning C structs and unions. <algnopt> can be one of: power, twobyte, or packed.
`arch=<option>`	Specifies the architecture on which the executable program will be run. The available options are: com—produces an object that contains instructions that will run on all the POWER and PowerPC hardware platforms. pwr—produces an object that contains instructions that will run on the POWER hardware platform. pwr2—produces an object that contains instructions that will run on the POWER2 hardware platform. pwrx—same as pwr2. ppc—produces an object that contains instructions that will run on any of the 32-bit PowerPC hardware platforms. The default is -qarch=com. If the -qarch option is specified without the -qtune=<option>, the compiler uses -qtune=pwr.
`attr=full`	Produces attribute listing (all names, whether referenced or not).
`chars=signed`	The data type char will be signed.

`datalocal=<name1>: <name2>: ...`	Specifies which data items are local. If no names are specified, all data items are assumed to be local.
`dataimported=<name1>: <name2>: ...`	Specifies which data items are imported. If no names are specified, all data items are assumed to be imported. This is the default.
`enum=<enumopt>`	Specifies whether minimum-sized enumerated types will be produced or not. <enumopt> can be either small or int. small denotes that either 1, 2, or 4 bytes of storage will be allocated for enum variables based on the range of the enum constants. int is the default and causes enum variables to be treated as though they were of type (signed) int.
`flag=<sev1>:<sev2>`	Specifies severity level of diagnostics to be reported in listing, <sev1>, and on screen, <sev2>.
`float=<opt1>:<opt2>: ...:<optN>`	The available options are: rndsngl—ensures strict adherence to IEEE standard. All operations on single-precision values produce results that remain in single precision. hssngl—rounds single-precision expressions only when the results are stored into REAL*4 memory locations. nans—detects conversion of single-precision NaNS to double-precision call checking. hsflt—never rounds single-precision expressions, and doesn't perform range checking for floating-point to integer conversions. nomaf—suppresses generation of multiply-add instructions. nofold—suppresses compile-time evaluation of constant floating-point expressions. rrm—specifies run-time rounding mode. Compiles with this option if the run-time rounding mode is rounded toward minus infinity; rounds toward positive infinity or not known. rsqrt—specifies whether a division by the result of a square root can be replaced with a multiply by the reciprocal of the square root. Default at -O2: -qfloat=norsqrt. Default at -O3: -qfloat=rsqrt. fltint—specifies whether range checking of floating-point to integer conversions is done. Default at -O2: -qfloat=nofltint. Default at -O3: -qfloat=fltint.
`flttrap=<opt1>:<opt2>: ...:<optN>`	Generates instructions to detect and trap floating-point. The available options are: overflow, underflow, zerodivide, invalid, inexact, enable, and imprecise.
`halt=<sev>`	Stops compiler after first phase if severity of errors detected equals or exceeds <sev>.
`ignprag=<pragval>`	Specifies the aliasing pragmas to be ignored. Used with #pragma disjoint and #pragma isolated_call. <pragval> can be disjoint, isolated, or all.
`initauto=<hh>`	Initialializes automatic storage to <hh>. <hh> is a hexadecimal value. This generates extra code and should only be used for error determination.
`isolated_call=<name1>: <name2>: ...`	Specifies that the calls to the functions listed have no side effects. <name1> and <name2> are function names. The user may specify as many function names as necessary.
`langlvl=<langlvl>`	Specifies language level to be used during compilation. <langlvl> can be ansi, saal2, saa, or extended.
`maxmem=<num>`	Limits the amount of memory used by space-intensive optimizations to <num>. <num> is specified in kilobytes.
`pgmsize=<p>`	Sets initial table size used by the compiler. <p> can be s for small or l for large.

`proclocal=<name1>: <name2>: ...`	Specifies which functions are local. If no filenames are specified, all invoked functions are assumed to be defined within the current file. The last explicit specification for a function takes precedence.
`procimported=<name1>: <name2>: ...`	Specifies which functions are imported. If no filenames are specified, all invoked functions are assumed to be defined outside the current file. The last explicit specification for a function takes precedence.
`procunknown=<name1>: <name2>: ...`	Specifies which functions are unknown to be local or imported. If no filenames are specified, all functions called are assumed to be unknown. This is the default when no user options are specified. The last explicit specification for a function takes precedence.
`spill=<size>`	Specifies the size of the register allocation spill area.
`tune=<option>`	Specifies the architecture system for which the executable program is optimized. The available options are: 601—produces an object optimized for all the PowerPC601 processors. pwr—produces an object optimized for the POWER hardware platform. pwr2—produces an object optimized for the POWER2 hardware platform. pwrx—same as pwr2.
`xref=full`	Produces cross-reference listing (all names, whether referenced or not).

As you can see, there are many options available with the standard XL C compiler system which comes with AIX. They will provide support for most of the compilation tasks you will need. There is included on the CD-ROM accompanying this book, however, an additional C compiler known as the GNU C compiler, which is also a best-of-breed compiler. It has certain advantages over the native XL C compiler, and you may want to investigate this before making a final decision about the C compiler to use.

3.5.1 Compiler modes including AIX 4.1 specific information

As mentioned in Sec. 3.5, there are a variety of modes in which the compiler can generate code. This is true for both XL C as well as C for AIX. A brief outline of the compiler mode issues is too important not to be mentioned in this book.

As outlined in the options portion of the XL C compiler in Sec. 3.5, the way to control the compiler output is through the use of the -qarch option. To recap, the available options are as follows:

`-qarch=pwr`	This generates code for the POWER instruction set. This will run on POWER, POWER2, and PowerPC-601 without the need for emulation. Software emulation may be required on the PowerPC 603 and 604.
`-qarch=pwr2`	This generates code for the POWER2 instruction set. This will run only on POWER2 if any of the unique POWER2 instructions are generated.
`-qarch=ppc`	This generates code for the PowerPC instruction set. This will run only on the PowerPC platform including 601, 603, and 604.
`-qarch=com`	In XL C this does not generate true common code, while on AIX 4.1 it does.

Each of these will support a particular compiler mode. While the first three support a specific platform or platforms, the fourth (or common mode) supports all hardware platforms. This is the recommended way of compiling with both XL C and C for AIX.

Common mode generates instructions which are common to all architectures. With XL C, it sometimes did not generate what is called millicode calls. These calls are specific low-level memory calls which allow for integer multiply, divide, and remainder operations. Because XL C did not generate these calls, you might run into compatibility problems and, therefore, not have true common mode. C for AIX generates millicode for all platforms and thus guarantees that code will be common and execute properly across all platforms. This means that you can generate a single binary image which will execute across all AIX platforms, regardless of the hardware platform.

For common mode to operate properly on AIX 3.2.5 with XL C, you need to ensure that you have installed the mandatory 8 PTFs that are required to include all relevant files. These are U432415, U432416, U432417, U432431, U432447, U432448, U432449, and U432450. See your system administrator for details on these.

Several steps must be followed to generate a common mode binary. They are:

1. Develop on AIX 3.2.5 (this is recommended to the support in the above mandatory eight PTFs).
2. Use the flags -qarch=com -qxflag=useabs on the compile line.
3. To link the application, use a special static library compiled in COMMON mode. This library is available from IBM as feature code 2504. You must then specify this library with the -I option on the compile line before all other system libraries.
4. Use the flag -bI:lowsys.exp where lowsys.exp is provided by the eight mandatory PTFs described earlier. This file contains symbols for the millicode that reside at the low memory locations. This allows for true common code to be generated.

Related to compiler modes are the tuning flags. You can tell the compiler to optimize common mode code for one type of hardware or another with the -qtune parameter. This will serve to increase performance if the application is run on the particular platform for which it was optimized. This will also cause the code to run significantly slower if run on a platform other than the one for which it was optimized. Keep this in mind if you are running across multiple architectures.

Finally, you can run in what is known as hybrid mode. This means that you can compile various routines within your application in more

than one mode and provide run-time testing to determine which one should be used. See the C for AIX guide or XL C guide for more details.

3.6 The Linkage Editor

The tool used to actually link all object files generated by the compiler is known as the linker, or linkage editor. This tool is responsible for combining object files, libraries, and import lists to create a final executable during the final phase of the compilation process as described earlier.

The basic format used by the linker is known as XCOFF (eXtended Common Object File Format). This is fast becoming the standard format for representing executable information under UNIX. The predecessor to XCOFF was COFF. XCOFF and COFF are very similar formats, the primary difference being that XCOFF supports dynamic linking.

You can invoke the linker directly with the ld command. As you can imagine, there are many options available to the ld command, some of which are shown below:

```
ld [-eLabel] [-Dnum] [-Hnum] [-KMm][-oname][-rs][-Snum][-Tnum][-vz]
[-Zstring]
[-Ldir][-ffile][-lkey...][-b option...] file...
```

where -eLabel makes the contents of the Label variable the entry point of the executable output file.

-Dnum makes the contents of the num variable the starting address for the initialized data of the ouput file.

-Hnum makes the value of the num variable the boundary to which the .text, .data, and .loader sections are aligned within the output file.

-K pads the header, .text, .data, and .loader segments of the output file to lie on page boundaries.

-M lists to the load map output file the names of all files and archive members processed to create the output file.

-m is the same as -M.

-oname names the output filename instead of the default a.out.

-r creates the output file even with unresolved symbols.

-s strips the symbol table, line number information, and relocation information from the output.

-Snum makes the value of num the maximum size of the user stack.

-Tnum makes the value of num the maximum size the user stack is allowed to grow.

-v is verbose mode.

-z is the same as -K.
-Zstring—prefix the standard library directory with string.
-L dir adds dir to the list of directories to be searched for library files.
-ffile takes the filenames to be processed from file.
-lkey processes the libKey.a file.
-b option is binder options.
file is the input file to be processed.

The only thing to watch out for relative to symbol definition and resolution is the use of reserved symbols. The XL C system generates reserved symbols which normally begin with _. Some of the more common ones are:

_text	Pointer to the first location of the program segment
_data	Pointer to the first location of the data segment
_etext	Pointer to the first location above the program segment
_edata	Pointer to the first location above the data segment
_end	Pointer to the first location above all data including dynamic and initialized data

This means that you must not use these symbols in your programs; otherwise you will have linker resolution problems.

Import and export files were discussed in the section relating to dynamic library and object creation; however, it is important to remember that the import and export files are used by the linker to dynamically "bind" objects from a shared environment into your executable. The binder is the software that actually does this. See the man page on ld for more information. See the earlier discussion for more details on their exact construction and syntax.

There was a simple example of usage of the linker command given in Sec. 3.2; this section provides several more examples in order to present a more thorough discussion of the linker. A simple example is as follows:

```
$ ld -T512 -estart -lc /lib/crt0.o -omain main.o sub1.o sub2.o
```

This will create an executable named main from three object files and a library named libc.a. The /lib/crt0.o is the run-time support object used by all C object code and needs to be included.

To review the creation of linking to a shared object, first create an export file as outlined above (let's call it common.exp) and use a command like:

```
$ ld -T512 -H512 -bM:SRE -lc -bE:common.exp -o common.o sub1.o
sub2.o sub3.o
```

This will create a shared object named common.o that can then be linked to other objects as shown next. You must first create an import file (let's call it common.imp) that follows the syntax outlined earlier. Once you have done this, you can issue a command like:

```
$ ld -T512 -H512 -lc -o main -bI:common.imp main.o -L ":"
```

Note that the ":" instructs the linker to include the current path in the search. This command generates an executable named main which can be invoked by typing:

```
$ ./main
```

There are many other ways to use the linker; it is often used as the last step in the compilation process and, therefore, is invoked automatically by the XL C compiler system.

3.6.1 C for AIX and AIX 4.1 changes to the linker/binder and loader

The binder has the following new features:

1. Increased performance. Compile times may be 2 to 5 times faster.
2. Smaller object files due to the inclusion of global scalars in the TOC entry.
3. Removed the TOC overflow problem by removing the TOC size limit of 64KB.
4. You can now use LIBPATH variable to override any previously defined -L options.

There are also a variety of new options as well as some obsoleted ones with C for AIX. See the man pages on the respective machines for more details.

There are also a variety of enhancements to AIX 4.1 and C for AIX relative to symmetric multiprocessing (SMP). These are beyond the scope of this book; however, it is safe to assume that all applications written for uniprocessor machines will run correctly on a multiprocessor machine. This is not always the case in the other direction. New technologies such as threads are making SMP a commercial reality. There is more on threads in Chap. 4.

Finally, the core file has changed in AIX 4.1 to support POSIX threads. This means that it won't be possible to analyze core dumps between AIX 3.2.*x* and AIX 4.1 operating systems. Keep this in mind if

you have any core dumps. You must analyze them on the same architecture on which they were generated.

3.7 Conclusion

This chapter has shown you some of the capabilities of the compilation system included with AIX 3.2.5 and available as a separate product with AIX 4.1. It is an extremely powerful and complex environment; however, once you understand its intricacies, you will find it extremely flexible and usable. IBM has led the industry in compiler technology for many years, and the XL C compiler subsystem along with C for AIX are the results.

While the GNU C compiler system is an alternative, consider the alternatives carefully before moving away from the native C compiler subsystem that is available for AIX.

Chapter

4

Native AIX Software Development Tools

4.1 Introduction

The previous chapter outlined the basic software development environment in AIX. This chapter presents several of the more important tools available to you in AIX that are related to software development and the free tools included with this book.

Much of the functionality discussed relating to the tools in this chapter is replicated by the tools discussed in Part 2 of this book. See these later chapters for more information about the tools included on the CD.

4.2 dbx

dbx is the interactive command line debugger that comes with most UNIX implementations. It is a symbolic debugger which allows you to:

Examine object and core files

Control the execution of an application

Set breakpoints and trace program execution and variables

Use symbolic variables and display them in their correct formats

Manipulate variables in virtual memory

Use several languages in the same executable and debug session

Languages most often supported are C, Fortran, Pascal, and COBOL. dbx is the Berkeley equivalent to sdb from AT&T. Most UNIX systems today use dbx as their standard debugger; sdb is not available on most machines today.

4.2.1 Using dbx

To invoke dbx use:

```
dbx [-a pid] [-c commandfile] [-d nesting] [-I dir] [-k] [-u] [-f]
[-r] [objectfile [corefile]]
```

where -a pid attaches the debug process to the process with process id of pid.
-c commandfile—dbx commands executed before beginning debug session.
-d nesting sets limit for nesting of program blocks; default is 25.
-I dir is directory to look in for associated source files; default is the current directory and directory where the executable is located.
-k maps memory addresses; useful for kernel debugging.
-u causes dbx to prepend symbols with an @ to avoid possible conflicting symbol names.
-f starts dbx reading only a minimum number of symbols to minimize start-up time and memory requirements (useful for large programs).
-r runs object file immediately; if program terminates successfully, dbx is exited.
objectfile specifies object file to debug.
corefile specifies core file to debug.

To use dbx, the programs must have been compiled with the -g option to generate symbol information which dbx uses.

When dbx starts, it checks for the existence of an initialization file .dbxinit in the current directory and the user's HOME directory. Any commands in the .dbxinit file are executed before the debugging session begins.

When you invoke dbx, you are placed in an interactive session from which you can issue commands and examine variables inside the program. For example, the following simple C program will be compiled and debugged:

```
$ cat test.c

main()
{
printf("this is a test of the debugger\n");
}

$ cc -g test.c
$ dbx a.out
Reading symbolic information...
Read 31 symbols
(dbx)
```

You are now at the dbx interactive prompt. From this point you can issue dbx commands and examine variables, change values, and run the program line by line. Note that a.out is the default object name from most UNIX compilers.

4.2.2 The dbx language

dbx commands are C-like in syntax and function. dbx works with expressions which consist of constants, operators, procedure calls, and variables. Some of the most important of these are:

Constants, which consist of constants declared within the program:

Character constants must be enclosed in single quotes
Octal format must be preceded with a 0.
Hex format must be preceded with a 0x.

Operators. The standard operators in most languages are:

+	Add
–	Subtract
*	Multiply
/	Divide
div	Remainder
<<	Bitwise shift left
>>	Bitwise shift right
&	Bitwise AND
|	Bitwise OR
~	Bitwise complement
&	Address and content of operator
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to
&&	Logical AND
||	Logical OR
sizeof(cast)	Size of variable or case
. field	Reference

4.2.3 Scope

Scope is a concept which defines the availability of constants and variables within procedures. The scope of variables is defaulted to within the current file and function. Values of variables are updated as functions are entered and exited. You can apply a specific scope with the file and func commands within dbx. Source files are expected to have the same name as the function with the proper language extension (.f, .c, etc.) to make it available to the compiler.

4.2.4 Running dbx

The basic dbx commands are:

`/ [regular expression]`	Searches forward in current source code for regular expression. Most often used to match strings (e.g., /string).
`?[regular expression]`	Searches backward in current source code. Opposite of /.
`alias [name ["command"]]`	Creates aliases for dbx commands for shorthand notation for commonly used commands. Note that the .dbxinit file is ideal for these commands. alias alone prints out all aliases.
`assign var=expression`	Assigns the value expression to the variable var; expression can be a string, logical type, or constant. See the examples for more information.
`call proc [params]`	Call executes the procedure specified by proc and passes parameters. Note that this procedure can be any standard procedure supported by the language (e.g., printf in the C language).
`case [default | mixed | lower | upper]`	Changes how dbx interprets symbols. Default is language dependent.
`catch [signum | signame]`	Sets a catch for the signal signum or signame before it reaches the program. If no parameters are used, all signals are trapped except SIGHUP, SIGCLD, SIGALARM, and SIGKILL.
`clear [line]`	Clears all breakpoints or only one on line if chosen. See set for more details; line can be either a line number (integer) or a filename followed by a colon.
`cont [signum | signame]`	Continues program from the current stopping point. If either signum or signame is included, the program continues as if it had been sent the appropriate signal contained in signum or signame.
`delete { number ... | all }`	Removes traces and stops from the current session. All traces and stops have an associated number which can be viewed with the status command and established with the trace and stop commands.
`detach [signum | signame]`	Continues execution but exits dbx. Useful if you have seen all you need in the debugger and simply want to finish the program.
`display [expr]`	Prints on the screen the value of expression, where expression is a regular expression.
`down [num]`	Moves the current functions down one level or num level in the call stack. This is relevant for scope and name resolution.

`dump [proc] [>file]`	Prints all variables local to the current procedure or named procedure. You can redirect your output to a specified file.
`edit [proc | file]`	Invokes an editor on the specific procedure or file. You can set the variable EDITOR to choose an editor other than vi.
`file [file]`	Changes the current source file to another. Simply type *file* to display the current source file.
`func [proc]`	Changes the current procedure to another. Simply type *func* to see your current context.
`help [cmd]`	Prints listing of dbx commands or more detailed description of command cmd.
`ignore [signum | signame]`	Ignores signals signum or signame sent to the current program.
`list [proc | line,line]`	Lists source lines either in the current procedure if no parameters are used, from proc if a procedure name is specified, or from line1 to line2 if a line expression is used. The default is 10 lines starting at the current line in the current procedure. The $ represents the current line, and you can use regular expressions to designate lines.
`multproc [on | off]`	Enables multiprocess debugging (not available on all dbx implementations).
`next [num]`	Executes one line or num lines jumping *over* function calls. Note that this means that a function call will be executed in its entireity and treated as a single line. See step for an alternative.
`print [expr]`	Prints the value of an expression. The expression can be any expression supported by the language used in the program.
`quit`	Exit dbx. Program execution is terminated.
`rerun [args]`	Begins execution again passing args as command line input parameters.
`return [proc]`	Continues execution until the procedure proc is reentered. If you don't specify a proc, you will execute until you leave the current procedure. This is not available on all implementations of dbx.
`run [args]`	Begins execution of the program, optionally passing args as command line arguments. The arguments should be entered exactly as they would on the command line.
`set var=expression`	Same as assign.
`sh [command]`	Executes a shell specified by the SHELL environmental variable. You can specify a command to execute within this shell with the optional command parameter. If you use command, when the command is finished, you are placed back in dbx.
`skip [num]`	Resumes execution skipping 1 or num breakpoints before honoring a breakpoint. This is not supported on all versions of dbx.
`source [filename]`	Executes dbx commands from the filename file.
`status [>file]`	Prints trace and breakpoint information which can be optionally placed in a file.
`step [num]`	Steps through one line of execution *into* calls. This means that if the next line of execution is a function call,

	you will stop at the first executable line in the function as a result of a step command. This is opposite of the next command.
`stop {var | [var] { at line | in proc}} [if condition]`	Sets breakpoints where program execution is halted. Execution is halted when: var—the variable var changes. at line—the source line is reached. in proc—the procedure is called. if condition—the condition is reached. dbx associates a number with each breakpoint. Use status to see these associations. You can use the delete function to remove them.
`trace [line | expression at line | proc | [var] | [at line | in proc]] [if condition]`	Prints tracing information specified on the dbx command line: at line—specifies a source line which contains the expression to be traced. if condition—specifies a condition for the trace to begin. in proc—specifies the procedure which contains the procedure or variable to be traced. See the examples sections for details.
`unalias name`	Removes the alias for name.
`unset var`	Removes the value of var.
`up [num]`	Moves the current function up the program stack. The default is 1.
`use [dir1 dir2 ...]`	Specifies which directories to use for source files separated by spaces; used by itself, it displays which directories are currently being searched.
`where [>file]`	Displays a list of active procedures. Output can be redirected to a file.
`which [name]`	Displays the fully qualified identifier name.
`whereis [name]`	Displays the fully qualified symbol name.
`whatis [name]`	Displays the declaration of name; name can be a function, procedure, variable, or constant.

Most dbx commands will print out the current status of associated variables or parameters if executed without any parameters (e.g., alias, case). There are also machine-level instructions which allow for low-level debugging at the machine instruction level. See your machine's specific dbx documentation since this is machine and debugger specific in many cases.

The above list is not all inclusive for all implementations of dbx, but it does include the majority of commands in dbx. If you use these commands fully, you will realize most of the power of dbx.

4.2.5 Example

The example below documents many of the dbx commands described above. The program test consists of three separate files: test.c, test1.c, and test2.c. All three files reside in the same directory, /tmp/book. The content of test.c is as follows:

```
main() {
int a=5;
printf("This is test and a is %d\n",a);
test1();
test2(a);
}
```

test1.c contains:

```
test1() {
printf("This is test1\n");
}
```

And test2.c contains:

```
test2(a) {
printf("This is test2 and a is %d\n",a);
}
```

The example is:

```
** Now compile the files to create a single executable a.out. **

% cc -g test.c test1.c test2.c
test.c:
test1.c:
test2.c:
Linking:

% dbx a.out # now invoke the debugging session

Reading symbolic information...
Read 80 symbols
(dbx) help /* printout help for SunOS dbx */
Command Summary

Execution and Tracing
catch clear cont delete ignore next rerun
run status step stop trace when

Displaying and Naming Data
assign call display down dump print set
set81 undisplay up whatis where whereis which

Accessing Source Files
cd edit file func list modules pwd
use / ?

Miscellaneous
alias dbxenv debug detach help kill make
quit setenv sh source

Machine Level
nexti stepi stopi tracei

The command "help <cmdname>' provides additional help for each
command
```

```
** Now help on a specific command. **
(dbx) help print
print <exp>, ... - Print the value of the expression(s) <exp>, ...
(dbx) print a
bad data address

** List status of all breakpoints, traces, etc. Note there are none
set yet. **

(dbx) status
(dbx) list /* list source code of current procedure */
2 int a=5;
3 printf("This is test and a is %d\n",a);
4 test1();
5 test2(a);
6 }
(dbx) step /* execute the first executable command but can't because
I have invoked dbx yet */ can't continue execution

** Note that you must issue the run command before you can invoke
any dbx execution commands since the run commands begins the execu-
tion of the program. What you typically do it set a breakpoint at
the first executable statement with the stop command and then type
run. **
(dbx) stop at 3
(dbx) run
Running: a.out
stop at 3
stopped in main at line 3 in file "test.c"
(dbx) list
3 printf("This is test and a is %d\n",a);
4 test1();
5 test2(a);
6 }
(dbx) status
(2) stop at "/tmp/book/test.c":3
(dbx) step
This is test and a is 5
stopped in main at line 4 in file "test.c"
4 test1();
(dbx) stop in test1

** Now set a breakpoint to stop at first executable line inside
test1 **

(4) stop in test1
(dbx) step /* step into test1; note next would have stepped over
test1 */
stopped in test1 at line 2 in file "test1.c"
(dbx) list /* lists source code inside current procedure */
2 printf("This is test1\n");
3 }
(dbx) trace test2
(5) trace test2 /* notify me whenever we enter test2 */
(dbx) status
(2) stop at "/tmp/book/test.c":3
(4) stop in test1
(5) trace test2
(dbx) delete stop in test2
(6) stop in test2 /* set a breakpoint at the beginning of test2 */
(dbx) status
(2) stop at "/tmp/book/test.c":3
```

```
(4) stop in test1
(5) trace test2
(6) stop in test2
(dbx) delete 6 /* remove the stop in test2 */
(dbx) status
(2) stop at "/tmp/book/test.c":3
(4) stop in test1
(5) trace test2
(dbx) continue /* whoops */
unrecognized command/syntax "continue"
(Type 'help' for help)
(dbx) cont /* continue on until next breakpoint or end of execution
*/
This is test1
calling test2(a = 5) from function main
This is test2 and a is 5
returning 5 from test2

execution completed, exit code is 1
program exited with 1
(dbx) rerun /* reexecute program maintaining all breakpoints, etc.
*/
Running: a.out
stopped in main at line 3 in file "test.c"
3 printf("This is test and a is %d\n",a);
(dbx) status
(2) stop at "/tmp/book/test.c":3
(4) stop in test1
(5) trace test2
(dbx) use /* which directory is everything in */
/tmp/book/
(dbx) file /* what source code file am I in now */
test.c
(dbx) func /* what is my current function name */
main
(dbx) list
4 test1();
5 test2(a);
6 }
(dbx) next /* step over test1 function call, see where we end up */
This is test and a is 5
stopped in main at line 4 in file "test.c"
4 test1();
(dbx) next
This is test1
stopped in main at line 5 in file "test.c"
5 test2(a);
(dbx) step
calling test2(a = 5) from function main
stopped in test2 at line 2 in file "test2.c"
(dbx) list
2 printf("This is test2 and a is %d\n",a);
3 }
(dbx) sh /* fork a shell, to get back to dbx type exit */
% ls
a.out test.c test1.c test2.c
book.script test.o test1.o test2.o
% exit
(dbx) where /* where is my current line position */
test2(a = 5), line 2 in "test2.c"
main(), line 5 in "test.c"
(dbx) quit /* quit dbx without finishing execution of my program */
```

The above example is all inclusive and demonstrates much of the power and functionality of the dbx debugger. While there are subtleties in this example which you may not understand, you can refer to it again as you learn more about AIX.

Another powerful use of dbx is to analyze where run-time errors are occurring in your programs. For example, let's assume you compile a program named test as follows:

```
$ cc -o test -g test.c
$ ./test
Bus Error - core dumped
```

When you attempt to run the application, you get a core dump. This means that you have had a run-time failure in your application. The system creates a file in the current directory named core which contains information relating to the most recent core dump. While you can go into dbx and step through the program line by line, you can get a quick notification of where the run-time error occurred by simply invoking dbx in the same directory with the command:

```
$ dbx test
dbx version 3.1
reading symbolic information...
[using memory image in core]
20 array[i]=0;
(dbx) quit
```

Note that this tells you that your run-time error occured on line 20 when you attempted to initialize an array variable. From here you can use dbx to step through your system to understand exactly what is causing the problem.

Finally, you can use dbx to attach to a running process and step through the system just as you could if you started the process with dbx. A simple example is:

```
$ ./test &
$ ps -u userid /* where userid is your userid*/
PID         TTY         TIME        COMMAND
666         hft/3       10:01       test
$ dbx -a 666
Waiting to attach to process 666 ...
Determining program name ...
Successfully attached to /home/kevin/test
dbx version 3.1
Type 'help' for help.
reading symbolic information ...
6 printf(array[i]);
(dbx)
```

This example demonstrates that you have initiated either a background process or a process in a different login session named test which has a process id (PID) of 666. You can attach to this process and manipulate it just as you would a process that you invoked from within dbx. This is a very powerful feature of dbx, and it allows you to manipulate daemons and other detached processes from a command line. Keep this feature in mind as you begin to develop in AIX.

4.2.6 dbx enhancements for AIX 4.1

While there were no dramatic changes to dbx with AIX 4.1, there were several enhancements worth mentioning. Some of the more important are:

1. Support for reduced size executables with the -g option was added.
2. dbx now supports full path information from the compiler. This makes it much easier to access the relevant source files.
3. Support for C type casting is now included in dbx.
4. Thread support is now included to support threads implementations in AIX 4.1.
5. dbx now supports long double types.
6. Enhanced support for multiprocessing.

One other note is that the xde graphical debugging program is gone in AIX 4.1.

4.2.7 Conclusion

This section has demonstrated a significant amount of the functionality of dbx. There are, however, more commands which can perform tasks which you may be interested in. See your man pages for your particular machine for more details. There is also a GNU version of dbx which provides enhanced functionality and commonality across platforms. The dbx session shown in this section was run from within a terminal window. There are tools which provide a more sophisticated interface to dbx; however, these are all changing in the near future as UNIX vendors change their interface, so they will not be documented here; however, see your local system documentation for more information on GUI-based dbx tools and use them just as you would use dbx as show above.

dbx is a powerful tool for software developers and maintainers. dbx in combination with other more sophisticated tools will help you to write and deliver better software.

4.3 lint

4.3.1 Introduction

The lint tool has been used for years to analyze C source code for syntax and possible run-time errors. lint can also check for nonportable and inefficient code. Some of the basic things you can do with it are:

Perform type-checking rules more strictly than with most compilers

Identify variable and function problems

Identify flow control problems

Identify inefficiencies in constructs

Identify unused and unreferenced code

Identify nonportable code

Identify code and library incompatibilities

4.3.2 Usage

The basic syntax is:

```
lint [-a] [-b] [-C][-c] [-h] [-lkey] [-n] [-olibrary] [-p] [-qDBCS]
[-u] [-v] [wclass [class ...]] [-x] [-MA] [-Ndnumber] [-Nlnumber]
[-Nnnumber] [-Ntnumber] [-Idir] [-Dname [=Def]] [-Uname] file ...
```

where -a suppresses messages concerning assignments of long variables to variables that are not defined as long.
-b suppresses messages about unreachable break statements.
-C specifies the use of C++ libraries.
-c produces a .ln file for every C file which can be used later by lint for more thorough analysis.
-h suppresses bug, style, and inefficiency checking.
-lKey includes a lint library for further cross checking. Key can be any of:
Key—includes the llib-lKey.ln lint library
m—includes the llib-lmath.ln lint library
dos—includes the llib-ldos.ln lint library
-n suppresses check for compatibility with standard and portable lint libraries.
-olibrary creates the llib-llibrary.ln library.
-p performs portability checks.
-qDBCS selects multibyte character set specified by locale.
-u suppresses messages about unused variables and functions.
-v suppresses unused function messages.
-wclass [class] specifies warning classes which determine what is reported. Some of the classes are:

a—non-ANSI features
c—comparison with unsigned values
d—declaration consistency
h—heuristic complaints
k—use for Kernighan and Ritchie (K&R) style C code
l—assignments of long variables to nonlong variables
n—null effect code
o—unknown order of evaluation
p—portability concerns
r—return statement consistency
u—proper usage of variables and functions
A—disables all warnings
C—constants occurring in conditional statements
D—external declarations never used
O—obselete features
P—function prototypes
R—unreachable code
S—storage capacity checks

-x suppresses messages about variables that have external declarations but were never used.
-MA enforces ANSI standards constructs in C code.
-Ndnumber changes table dimension to number.
-Nlnumber changes number of type nodes.
-Nnnumber changes symbol table size to number.
-Ntnumber changes tree node numbers to number.
-Idir adds dir to directories to search for #include files.
-Dname=Def is a macro definition similar to that used by cpp.
-Uname removes definition of name where name is a symbol used by the program.
file is any number of files to scan with lint.

lint has been in use for a long time and has a history of support for the K&R style of C code. It has only recently has begun to support ANSI standard C. Keep this heritage in mind when you are using lint to analyze code.

There are a number of strings you can place within your source code to control lint's behavior. They are beyond the scope of this chapter. See other lint documentation for more information on these commands.

4.3.3 Examples

To check a simple program for syntax errors, issue the command:

```
$ lint kevin.c
```

To check a series of files, you should first run each file through with the -c option, which produces a .ln file. After performing this operation on each file, run lint on the result with the appropriate -l options to generate lint statements that reference the appropriate file. If you don't use this method, you will get lint messages from unknown file locations.

```
$ lint -c file1.c
$ lint -c file2.c
$ lint -lfile1 -lfile2 file1
```

Each lint -c command generates a file with a .ln extension which is a lint library. This is then cross-referenced in the last command and will produce error messages which reference the appropriate file. This is particularly useful for makefiles since you can lint only those files that have changed and can issue the appropriate lint command with the correct -l options to regenerate the executable.

The lint executable checks your C source file against a variety of data files it uses to store standard syntax and rules. The basic files under AIX are listed in Table 4.1.

4.3.4 Conclusion

lint is a very powerful analysis tool for C code. Use this before compilation to check for inconsistencies and for syntax and run-time errors. You can also search for unused and inefficient code before you waste

TABLE 4.1 AIX lint Standard Libraries

lint Library Name	Contents
/usr/ccs/lib/llib-lansi	Declarations of standard ANSI functions (source)
/usr/ccs/lib/llib-lansi.ln	Declarations of standard ANSI functions (binary)
/usr/ccs/lib/llib-lc	Declarations for standard functions (source)
/usr/ccs/lib/llib-lc.ln	Declarations for standard functions (binary)
/usr/ccs/lib/llib-lcrses	Declarations for curses functions (source)
/usr/ccs/lib/llib-lcrses.ln	Declarations for curses functions (binary)
/usr/ccs/lib/llib-lm	Declarations for standard math functions (source)
/usr/ccs/lib/llib-lm	Declarations for standard math functions (binary)
/usr/ccs/llib-port	Declarations of portable functions (source)
/usr/ccs/llib-port.ln	Declarations of portable functions (binary)
/usr/ccs/xlC/lib	Directory containing C++ libraries
/var/tmp/*lint*	Temporary files

time with more sophisticated performance analysis tools. lint is one of the most powerful tools available on the AIX platform for code analysis and design.

4.4 prof and gprof

4.4.1 Introduction

Profiling consists of code analysis to understand where you are spending most of your resources, including CPU time, I/O, and memory. With profiling tools you can study how your program behaves and where it is using the most resources. Once you have found the "hot spots" in your code where it spends most of its time, you can focus on fine tuning these areas to increase the performance of your overall system.

The general profiling and application-tuning utilities available with AIX are prof and gprof. While these tools do not offer the functionality of many performance and profiling tools that you can purchase, they do offer basic capabilities which will assist you in monitoring and analyzing the hot spots and other problems with your code.

To take full advantage of the profiling, you must compile your code with the -p option for use with the prof command and with -pg for use with the gprof command. See the sections below for more details.

Profiling your code will provide information on the percentage of time spent in each function, the number of times a particular function was called, and the number of milliseconds spent within each function. While the granularity of the statistics made available is not high, it will provide you with enough information to structure your code differently if necessary.

4.4.2 Usage

prof usage. The basic syntax for the prof command is:

```
prof [-t | -c | -a | -n] [-o | -x] [-g] [-z] [-h] [-s] [-S] [-v]
[-L path] [prog] [-m file...]
```

where -t sorts by decreasing percentage of total time (default).
-c sorts by decreasing number of calls.
-a sorts by increasing symbol address.
-n sorts by symbol name.
-o displays addresses in octal.
-x displays addresses in hex.
-g includes nonglobal symbols.
-z includes all symbols, even those not referenced or executed.
-h suppresses default heading.
-s produces a summary file in mon.sum.

-S displays statistics on standard error.
-v displays output graphically on standard output.
-L path uses alternate path for shared libraries.
prog is the program to execute.
-m file takes profiling data from file instead of from mon.out.

To use prof effectively, you should first compile your codes with the -p option and execute normally. This produces a file named mon.out by default which contains information on that particular iteration of the code. Once that has been run, you would issue a command like:

```
$ prof -t
Name        %Time       Seconds     Cumsecs     #Calls      msec/call
.printf     52.0        0.02        0.02        6           2.
.main       42.0        0.02        0.04        2           1.
sub1        8.0         0.01        0.05        1           1.
```

As you can see, it produces a decreasing listing of times spent within a particular function call. The columns are self-explanatory and consist of percentage of time spent in each routine, total section in each routine, accumulated time for the overall program, the number of calls from each routine, and the milliseconds per call for each subroutine or function. This will give you a good estimate of how much time your system is spending in each routine as a percentage of total execution time. You can issue the command:

```
$ prof -L/usr/share/lib kevin.out -Mkevin.mon
```

It will generate information using shared library files contained in /usr/share/lib and will use the executable kevin.out and the monitor data from the file kevin.mon instead of from the default mon.out.

gprof usage. The basic syntax for the gprof command is:

```
/usr/ucb/gprof [-b] [-e name] [-E name] [-f name] [-F name]
[-L path] [-s] [-z] [a.out [gmon.out ...]]
```

where -b suppresses field descriptions.
-e name suppresses graph profile entry for name and all of its descendants.
-E name suppresses graph profile entry, time spent, and percentage time information for name.
-f name displays graph profile entry for name and its descendants.
-F name displays graph profile entry and time and percentage entries for name and its descendants.

-L path uses path for locating shared libraries instead of default.
-s produces gmon.sum, which sums statistics for multiple gprof executions.
-z displays functions that have zero execution times.
a.out is default executable name.
gmon.out is default gprof statistics file.

The basic operation of gprof is the same as prof. After compilation of the source code with the -pg option, you invoke the resulting executable as you normally would. This results in a file named gmon.out, which contains information which is used by gprof. Once you have collected the information by running your program, use a command like:

```
$ gprof
# gprof
# @(#)64 1.4 com/cmd/stat/gprof/gprof.callg, bos, bos320 7/31/91
18:48:5
#
# COMPONENT_NAME: (CMDSTAT) gprof
#
# FUNCTIONS: N/A
#
# ORIGINS: 27
#
# (C) COPYRIGHT International Business Machines Corp. 1989
# All Rights Reserved
#
# US Government Users Restricted Rights - Use, duplication or
# disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

call graph profile:
The sum of self and descendents is the major sort for this listing.

function entries:

index the index of the function in the call graph listing, as an aid
to locating it (see below).

etc...

0.00 0.00 13/13 ._doprnt [5]
[1] 0.0 0.00 0.00 13 .fwrite [1]
0.00 0.00 13/13 .memchr [2]
0.00 0.00 3/7 ._xflsbuf [3]
0.00 0.00 1/2 ._wrtchk [20]
-----------------------------------------------
0.00 0.00 13/13 .fwrite [1]
[2] 0.0 0.00 0.00 13 .memchr [2]
-----------------------------------------------
0.00 0.00 1/7 .fflush [22]
0.00 0.00 3/7 ._flsbuf [7]
0.00 0.00 3/7 .fwrite [1]
[3] 0.0 0.00 0.00 7 ._xflsbuf [3]
0.00 0.00 7/7 .write [4]
-----------------------------------------------
```

```
etc.

# @(#)65 1.4 com/cmd/stat/gprof/gprof.flat, bos, bos320 7/31/91
18:49:52
#
# COMPONENT_NAME: (CMDSTAT) gprof
#
# FUNCTIONS: N/A
#
# ORIGINS: 27
#
# (C) COPYRIGHT International Business Machines Corp. 1989
# All Rights Reserved
etc.

% cumulative self self total
time seconds seconds calls ms/call ms/call name
0.0 0.00 0.00 13 0.00 0.00 .fwrite [1]
0.0 0.00 0.00 13 0.00 0.00 .memchr [2]
```

This will produce several outputs. The first is very similar to that produced by prof, including function times as a percentage of total execution time, number of times the functions are called, and the total execution times of each. Times are then propagated to a call graph as illustrated above. The second piece of output includes call graph execution times including time distribution to the descendants. Finally, cycles are shown, including an entry for the cycle as a whole and a listing of the members of the cycle and their individual cycle and call count times. The above is a very limited presentation of the actual output of the gprof command. Run some examples on your local machine for more information.

4.4.3 Conclusion

prof and gprof provide basic profiling capabilities which allow for a certain level analysis of code to occur, including performance and analysis information. By using these tools, you can better understand the execution characteristics of your code and thereby perform the appropriate actions on it to enhance performance and any other desired characteristics.

There are other commercial tools available which do far more than prof and gprof, but these tools do provide the basics that you need to tune your code effectively and efficiently.

4.5 ar

4.5.1 Introduction

The ar command is used to create and manipulate archive files. These are libraries of files which are typically used for the link process. Files

are created by a compiler into a format known as the object format and can then be stored as members in an archive file. These members are then used by the link editor to generate a final executable.

4.5.2 Usage

```
ar [c][l][o][s][v]{m [a|b|] position | r [a|b|i|u] position |
{d|h|q|t|w|x}} archivename [membername ...]
```

where c suppresses normal creation messages.
l places temporary files in current directory instead of in default /tmp.
o sequentially orders and compresses archive file.
s regenerates symbol table.
v is verbose mode.
m moves members within an archive:
a position—move to position following position
b position—move to position preceding position
i position—same as b position
r replaces members within an archive; a position, b position, and i position are the same as the m option.
d deletes member from archive.
h changes modification times of members to current date and time.
q displays contents of named members or entire archive if no member is specified.
t displays table of contents.
w displays symbol table.
x extracts members to current directory.
archivename is the name of archive library.
membername ... is the name or names of members to be manipulated.

The archive library consists of members generated by a compiler and a symbol table which is used by the link editor to create an executable. Most operations which affect members cause a regeneraton of the symbol table; however, this is not always the case. See the examples below for more information.

The basic options for the ar command must be used as described above. You must select one of closv and one of dhqtwx. The rest are optional and depend on the other options chosen. Keep in mind that the options must be placed seqentially on the command line with no intervening spaces.

4.5.3 Linking

The linking process generates a single executable file from a series of object files generated by a compiler. Most linkage editors are contained within the command used to invoke the compiler. For example, the cc command by default invokes both the compiler and the linkage editor. The same holds true for the f77 command.

When the linkage editor examines the link statement, it performs a single pass through all referenced files and archives to generate an executable file. The first discovered reference is used to build the executable. This means that if you have multiple references to the same member or object filename, the linkage editor will use the first. This implies that the order of members within the archive library is important to determine the final executable contents. Keep this in mind when generating the archive library and using the position commands such as m and r to move members within an archive. See the examples below for more information. Also see Sec. 3.6 on the ld command which describes the linkage editor in more detail.

Suffice it to say that most developers use archive libraries as function or as subprogram libraries, especially when the software systems get large. This provides an easy way to track and maintain groups of functions or subprograms.

4.5.4 Examples

These examples will assume the existence of four object files (member1.o, member2.o, member3.o, and member4.o) in a single directory.

To generate an archive library from these files, use the command:

```
$ ar vq members.a member1.o member2.o member3.o member4.o
```

This will create an archive file named members.a which contains four members named member1.o through member4.o.

If the archive file members.a already exists, it will add these four members to the end of the archive without checking for previous members of the same member name. This is important since the linkage editor will use the first occurrence of the member name to generate the executable. Keep this in mind as you create more archive libraries. It is generally *not* a good idea to create equivalent member names within an archive library; however, if you know what you are doing, this can be a powerful technique.

To view the results of your archive creation, issue the command:

```
$ar vt members.a
rw-r--r--  0/0    4997 May 01 10:14 1993 member1.o
rw-r--r--  0/0    5121 May 01 10:14 1993 member2.o
```

```
rw-r--r--  0/0    4030  May 01 10:14 1993 member3.o
rw-r--r--  0/0    10939 May 01 10:14 1993 member4.o
```

This generates a table of contents which is similar to ls -l.

To replace a member, use the command:

```
$ ar vr members.a member1.o
```

You can use the positioning command to affect the order of members in the archive. To add the contents of a modified members1.o file, you can use the command:

```
$ ar vq members.a member1.o
```

Note that this command creates a duplicate member. The results of the table of contents command show the following:

```
$ar vt members.a
rw-r--r--  0/0    4997  May 01 10:14 1993 member1.o
rw-r--r--  0/0    5121  May 01 10:14 1993 member2.o
rw-r--r--  0/0    4030  May 01 10:14 1993 member3.o
rw-r--r--  0/0    10939 May 01 10:14 1993 member4.o
rw-r--r--  0/0    4128  May 02 10:24 1993 member1.o
```

This is dangerous; however, it does provide certain functionality that you may need. You can position files within the archive with a command like:

```
$ ar vma member3.o members.a member2.o
```

This moves the member member2.o to follow member3.o:

```
$ar vt members.a
rw-r--r--  0/0    4997  May 01 10:14 1993 member1.o
rw-r--r--  0/0    4030  May 01 10:14 1993 member3.o
rw-r--r--  0/0    5121  May 01 10:14 1993 member2.o
rw-r--r--  0/0    10939 May 01 10:14 1993 member4.o
rw-r--r--  0/0    4128  May 02 10:24 1993 member1.o
```

This command moves the member member2.o to follow the member member3.o. You may want to do this to place a global symbol contained in member3.o before the same global symbol contained in member2.o. This is related again to linkage editor resolution requirements.

To extract a member, use the command:

```
$ ar vx members.a member2.o
```

This will place the contents of member2.o in a file named member2.o in the current working directory. Because of ar's use of standard I/O, you

can use redirection and piping as you would with most other AIX commands.

For example, to rename the results of the above extraction to a file named something other than member2.o, use:

```
$ar vx members.a member2.o > cmember2.o
```

This will create a file named cember2.o that contains the contents of the member member2.o.

You can delete a member in an archive with the command:

```
$ ar vd members.o member4.o
```

As you have made changes to the archive, its structure and order have changed. Because of the structure of the archive, it may have unused space and inefficiencies within itself. To reorder and compress the archive, use the command:

```
$ ar vo members.a
```

This sequentially orders and compresses the members.a archive. This is particularly useful after a number of delete operations since these often don't compress the archive file as efficiently as possible.

Finally, you can use the strip command (see information on the strip command) to remove symbol and other information, and this is related to the ar command. You may want to strip the archive library to remove many deleted symbols from deleted members. After you strip the archive library, rebuild the symbol tables with the command:

```
$ ar vs members.a
```

This will generate a clean up-to-date copy of the global symbols contained in the members within the archive. To view the new symbol table, use the command:

```
$ ar vw members.a
```

There are other commands related to archives such as strip and ld. See Secs. 4.7 and 3.6 for more details about strip and ld and more examples of how to use archive libraries.

4.5.6 Conclusion

The ar command is a powerful command that creates and manipulates archive libraries. These libraries can help you organize your develop-

ment effort and control the generation of executables. There are special provisions in make and in the linkage editor which take advantage of archive libraries. UNIX power developers take full advantage of the ar facility.

4.6 nm

4.6.1 Introduction

The nm facility generates a listing of the symbols in an object file. The file can be a simple object file, an executable file, or an archive file. Each symbol is preceded by a value which defines the characteristics of the symbol itself.

There are two versions of the nm command: Berkeley and AT&T. They use different syntax but perform the same basic tasks. Keep this in mind and examine the manual pages for AIX.

AIX is largely a System V-based UNIX operating system and, therefore, the nm command follows the System V option conventions. The syntax for nm is:

```
nm [-C] [-O] [-T] [-e] [-f] [-h] [-r] [-u] [-n | -v] [-o | -d | -x]
[file ...]
```

where -C suppresses the mangling of C++ names.
-O displays file or archive name with each symbol rather than once.
-T truncates symbol names as necessary.
-e displays static and external symbols.
-f displays all symbols.
-h does not display header information.
-r displays in reverse order.
-u displays undefined symbols.
-n displays external symbols ordered by name; use with the -e option.
-v displays external symbols ordered by value; use with the -e option.
-o displays values in octal.
-d displays values in decimal.
-x displays values in hex.
file ... is one or more files to operate on.

An example of a simple nm command is:

```
% nm bin/kermit | more
000227a4 d _ATT7300
000225c4 d _CERMETEK
```

```
0002277c d _CONCORD
000222f0 D _DELCMD
000225ec d _DF03
00022614 d _DF100
0002263c d _DF200
000222fc D _DIRCMD
00022d4c d _EXP_ALRM
000228ac d _F_reason
00022664 d _GDC
0002268c d _HAYES
000226b4 d _PENRIL
000222f4 D _PWDCMD
000226dc d _RACAL
00022304 D _SPACM2
00022300 D _SPACMD
000222f8 D _TYPCMD
00022704 d _UNKNOWN
0002272c d _USROBOT
00022754 d _VENTEL
00022308 D _WHOCMD
```

The characters preceding the symbol name designate the following:

A—absolute variable

B—BSS segment symbol

D—data segment variable

T—text segment symbol

U—undefined symbol

f—file name symbol

-—debugger symbol

Symbol information is sorted alphabetically by symbol.

The AIX variant of this output is slightly different but represents essentially the same type of information. With the AIX variation, the variable types and contents are generally described in a little more detail. Example output might look like:

```
$ nm transform
$ nm transform | more
Symbols from transform:

Name Value Class Type Size Line Section

.__start | 512|extern| | | |.text
__start | 648|extern| | | |.data
_adata | 216|unamex| | | |.data
TOC | 656|unamex| | | |.data
_adata | 692|unamex| | | |.data
errno | 696|unamex| | | |.data
...
```

The nm command is a very useful tool to understand, and it determines the scope and function of all variables in an object or executable file. For example, when you have an executable and you would like to understand which symbols are defined where, you can use the nm command to determine this. nm is also useful to better understand the structure of code that exists without source code. nm provides access to some information on the structure and content of object and executable files. This is one of the first things you may want to do when taking a look at someone else's code.

To display symbol sizes and values in octal and sort by value, use:

```
$ nm -eov kevin.out
```

To display external symbols, use:

```
$ nm -e kevin.out
```

4.6.2 Berkeley usage

AIX contains the Berkeley version of the nm command in the /usr/ucb directory. To invoke it, type:

```
$ /usr/ucb/nm
```

See the man page for more information on the syntax of this command.

4.6.3 Conclusion

The nm command is frequently used by power users to provide information on the structure and content of object and executable files. Both versions of the command (Berkeley and AT&T) provide similar functionality and can be used by anyone to learn more about binary files.

4.7 strip

4.7.1 Introduction

The strip command removes symbols and relocation information from object files. Symbols and relocation information are placed in the executable for linking and debugging purposes. Many of the tools you use to debug your code as well as to link and compile your code rely on this information in the executable file. Once you have finished debugging, however, you may want to remove this for a variety of reasons.

Removing this information is useful when you want to decrease the size of your executable and remove all unnecessary information from the binary file before using it in production mode. By doing this, you

shrink all resources required to run it, including memory, disk, and CPU. This is often used by more advanced AIX developers, particularly in real-time system development and embedded control systems where resources are tight. It is generally used to make object files and subsequent executables smaller without sacrificing performance. This is generally used after a program has been completely debugged and is ready for distribution and use.

4.7.2 Usage

The basic syntax for strip is:

```
strip [-V][-r][-l]]|-x [-l]|[-t | -H]file ...
```

where -V displays strip version number.
-r removes all symbol table information except external and static symbols. Does not remove relocation information.
-l removes line number information.
-x removes symbol table information except external and static symbols. Does remove relocation information.
-t removes symbol information except function symbols or line number information.
file ... is one or more object code files.

Once you have compiled a file and debugged the application, you can issue the strip command on the resultant executable to remove all symbol table information and relocation bits. These are only used by the debugger and linker and are not relevant to the execution process. This allows you to minimize the size of your executable while not effecting the execution of your program.

A simple example is:

```
$ strip file.o file1.o file2.o
```

Once you have stripped the object files, you can link as you normally would with a ld or some other precompiler command such as f77 or cc.

4.7.3 Conclusion

The strip command is useful when you want to minimize the size of your executable and, therefore, minimize disk space and memory requirements. You can invoke the strip functionality either with the strip command or with the -s option on the link (ld) command. See the ld command (Sec. 3.6) for more information.

strip provides the flexibility to minimize executable size while not affecting the execution process of your program.

4.8 The r Commands

4.8.1 Introduction

The r commands consist of several commands which begin with an r. The r designates remote. These commands allow you to emulate local commands on a remote machine. Basic examples of the r commands are:

rsh

rcp

rlogin

These commands were traditionally shipped with the Berkeley derivative of the UNIX operating system. Because of this, most versions of UNIX, other than standard System V, came with the r commands. These commands allow you to execute the cp, sh, and login commands remotely without requiring a password. Because of this they are often seen as inferior to the standard FTP and telnet sorts of operations. However, since these commands are so widely used, they are still included with almost every UNIX operating system shipping today.

4.8.2 Usage

Security and the r commands. The r commands use three files to perform user authentication on the remote machine. The first is the global r security file called /etc/hosts.equiv. This file contains a global mapping of hostnames and usernames supported for remote access. If the remote machines contains a /etc/hosts.equiv file which equates to the local and remote hosts and usernames, you will be allowed access if the remote machine is in the /etc/hosts file and you have an account in both /etc/passwd files.

The basic syntax of the /etc/hosts.equiv file is:

```
hostname username
hostname username username
...
```

where hostname is the hostname of a particular machine on the network and username is the name of a user who you want to allow access on the local machine. For example, if you place the line:

```
devtech kevin
```

in the /etc/hosts.equiv file on a machine named ibmgod and issue a rlogin, rsh, or rcp command from devtech to ibmgod and you are the account kevin, you will permitted access. Note that the username specifies that you can share usernames between machines. You can allow all users with entries in the /etc/passwd machine on both machines access with the line:

```
devtech +
```

in the /etc/hosts.equiv. This says that all users with matching id's on ibmgod and devtech will be allowed access to their exact account on the ibmgod machine from devtech. If you placed this line in the /etc/hosts.equiv file on ibmgod and executed the command:

```
$ rsh ibmgod ls
```

from the devtech machine, you would be logged on to ibmgod, and the command ls would be executed on your HOME directory just as if you had logged on and issued the ls command. Note that you must have the same account on both machines (but not the same password), and there must be a mapping in the /etc/hosts.equiv file for this to work correctly.

You can also allow only specific other accounts access to your account with the r commands by placing their account names and machine specifically in the .rhosts file. An example file might be:

```
ibmgod root kevin
devtech kevin glen
pegasus gch psm glen
```

This says that the root and kevin accounts can issue the r command in the current account from ibmgod, kevin and glen can execute commands from devtech, and gch, psm and glen can execute commands from pegasus. This allows you to pick and choose who you give "passwordless" access to your account to. This is key to successfully controlling access to your account.

Given all of this discussion about the /etc/hosts.equiv file, it is important to note that it is generally not a good idea to use it. The better solution is to create a file named .rhosts in your HOME directory that contains the mapping information exactly as described in the /etc/hosts.equiv file but applies only to your account. This file is consulted after the /etc/hosts.equiv file to see if access is allowed. The syntax of the .rhosts file is exactly the same as that of /etc/hosts.equiv. For example, if you wanted to allow access to your kevin account on devtech from your kevin account on ibmgod, you would create the following $HOME/.rhosts file on devtech:

```
ibmgod kevin
```

This would specifically allow the kevin account on ibmgod to access the kevin account on devtech without requiring a password for the r commands.

You can also allow others to access your account from this file by creating a .rhosts file like:

```
ibmgod
```

This will allow all users on ibmgod to access the kevin account on devtech without requiring a password. This is obviously a bit of a security issue and should be avoided if possible.

It is generally not a good idea to allow root to access other machines without a password. If you place the line:

```
ibmgod
```

in the /etc/hosts.equiv or /.rhosts file on devtech, root from ibmgod now has access to root on devtech without a password. Even if you maintain both systems, it is generally not a good idea to do this and should be avoided.

While there are a variety of security holes and problems associated with this methodology, it is generally used and is exceedingly powerful when it comes to saving time moving files and information around in a network.

The r commands and login scripts. People often experience strange problems with the r commands which cause rcp to fail and rsh to work intermittently. Often this is caused by something in their login scripts issuing output to standard output. If your login scripts (.login, .profile,.cshrc, etc.) issue output to standard output, they may confuse the r commands and cause either intermittent or complete failure of the r command itself. Check your login scripts and ensure that you do not issue any reads or writes from within them, or code them such that they check your terminal type to ensure that they are local when executing them. If you are not local, you should not execute any I/O since you may have problems wth your r command execution.

The rlogin command. The rlogin command allows you to log on to a remote machine without typing a password. The basic syntax is:

```
rlogin hostname [-e character] [-l username] [-8]
```

where hostname is the remote machine hostname.
-e character changes the escape character.
-l username specifies a username which can be other than your current username.
-8 establishes an 8-bit data path instead of the usual 7.

rlogin provides a virtual terminal session into a remote computer. From this you can execute applications and run just as if you were logged on to the remote machine directly. You can use the -l option to specify an account other than the matching account for your current login id on the remote machine.

The standard escape character, unless modified with the -e flag, is tilde (\~). This means that you can escape back to the local machine by entering the \~. sequence. The period (.) designates that you want to end the remote session.

The rsh command. The rsh command allows you to execute remote commands on machines without issuing a password. Using the /etc/hosts.equiv and .rhosts file as described above, you can transparently access and run remote commands and display the output locally.

The basic syntax of the rsh command is:

```
rsh hostname [-l username] [-n] [command]
```

where hostname is the remote hostname you wish to connect to.
-l username allows you to specify a username other than your current one.
-n sends input to the null device (/dev/null).
command is the command to execute on the remote machine.

rsh is used to execute remote commands from the local command line. A simple example is:

```
$ rsh ibmgod ls
```

This, as described earlier, will log you on to ibmgod with your current userid and issue the ls command in your home directory. You can execute any command from this rsh command. Standard I/O is mapped and appears local as you expect it to if you were executing the command locally.

A more interesting example is:

```
# rsh -l kevin devtech ls;echo $PATH;cat .profile
```

If you execute this as root (denoted by the #) from the ibmgod machine, it will execute all three commands on the devtech machine as userid

kevin. This assumes that you have specifically allowed access to root from ibmgod access to the kevin account as described above. If you have not allowed specific access, you will get an access denied error message.

Note also that sometimes the -l username option is after the hostname. See your local system for documentation on the exact syntax for your machine.

Another interesting example of using the r commands to make moving data between machines more interesting is the following:

```
$ rsh ibmgoc cat file1 ">>" file2
```

This will append the contents of the remote file file1 to the remote file file2. Note that you must include any shell metacharacters (in this case the >>) to prevent the shell from interpreting them.

The rcp command. The rcp command allows you to do a remote copy of a file without requiring a password. The basic syntax is:

```
rcp[-p] [-r] file1 file2
```

where -r recursively copies any directories underneath the current one.
-p preserves the modification times and modes of the original files.
file1 is the file to copy from.
file2 is the file to copy to.

The syntax of the file is:

```
[username@]hostname:filename
```

where username is the name of the remote user (default is your current userid).
hostname is the name of the remote host.
filename is the filename either fully qualified or given a relative path from the HOME directory of the user.

Note that the username is not required and will default to your current userid on the local machine. The colon (:) is what tells rcp that you are manipulating a remote file. This provides you with the ability to transfer a file without requiring a password like FTP does. This is very commonly used by users of several machines in a network and is definitely a time-saving feature of UNIX.

A simple rcp command might look like:

```
$ rcp devtech:/tmp/file /tmp/file
```

If executed on ibmgod, this command will look for a file named /tmp/file on the remote machine devtech and attempt to copy it to the ibmgod machine and place it in /tmp/file. You can use a command like:

```
$ rcp devtech:.rhosts .rhosts
```

which will copy the remote machine's (devtech) .rhosts file in your HOME directory to your curent directory on your local machine.

The filename can be a directory if you wish to place the file in a directory. This is particularly useful if you are copying a group of files. A simple example might be:

```
$ rcp file1 file2 file3 ibmgod:
```

This will copy three files and name them file1, file2, and file3 in your HOME directory on ibmgod. You can use wildcards to match filenames as your normally would with UNIX. A simple example is:

```
$ rcp devtech:"*.txt" textfiles
```

Note that textfiles must be a directory. In fact, anytime you copy multiple files, you must use a directory; otherwise what is the filename of the three files? Note also that if the wildcards are to be expanded on the remote systems, you must enclose them in quotes to prevent the local shell from interpreting them before passing them to the r command.

A recursive copy looks like:

```
$ rcp -r devtech:prog prog
```

This will copy all files, recursively, from devtech and the subdirectory prog to the current directory prog. Note also that symbolic links are not supported in this environment and actual copies of the files will be made. Therefore, if you have symbolic links in some directories which you are copying, you will need more disk space. If you are interested in preserving the exact structure of the data, you need to issue a command like:

```
$ tar cvf - test | rsh devtech tar xBf -
```

This will copy the current directory structure test to a remote machine named devtech and place it in a test subdirectory in your account on devtech. Note that this is a very powerful way of moving files around in your network.

The final example of using r commands is used to access and control

remote devices. To retrieve a tar file from a remote tape device on devtech from ibmgod, you might use a command like:

```
$ rsh devtech dd if=/dev/rmt0 obs=16b | tar xvfBb - 16
```

This will dump (dd) the files from the tape drive (/dev/rmt0) with a blocking size of 16 to standard output. The tar command will take input from standard input (-) and place in on the local disk in the tar format in which it is received.

To copy a file to a remote tape device, you might use a command like:

```
$ rsh tar cvfb - 16 file1 file2 group1 | rsh devtech dd of=/dev/rmt0
ibs=16b
```

This will take file1 and file2 files as well as the contents of the group1 directory and place them in a tar file on the remote tape device /dev/rmt0 on the remote machine devtech.

These kinds of tools represent the kind of tricks you can perform with UNIX and illustrate some of the kinds of things you can do without writing a single line of code.

4.8.3 Conclusion

The r commands are often used by people who want to increase their effectiveness with AIX. Because of the security implications of using the r commands, however, it is important that you understand exactly what you are doing and ensure that you are not opening up security holes in your network. Keep this in mind as you begin to look at these tools more carefully.

4.9 install

4.9.1 Introduction

The install command is used by many software packages to install into a particular directory or set of directories. It is often used by free software tools in the build process to place files in particular directories. Because of this, it is included in this section so that you will understand what it does later on in this book.

4.9.2 Usage

There are two styles of syntax for the install command:

```
install [-c dir] [-f dir] [-imosS] [-M mode] [-O owner] [-G group]
[-n dir] file [dir...]
```

where -c dir installs file in dira only if it does not previously exist in dir.
-f dir forces installation of file even if it already exists in dir.
-i ignores default directory list and uses only command line directories.
-m moves the file instead of copying it.
-M mode specifies mode of destination file.
-o saves copy of file as OLDfile in the same directory.
-O owner specifies a different final owner than your id.
-G group specifies a different group for the installed file.
-n dir installs file in dir if it is not in any of the searched directories.
-s displays error messages only.
-S strips binary after installation (see strip for more information).
file is file to be moved.
dir is directory in which to move the file.

install searches the default directories /usr/bin, /etc/ and /usr/lib in that order for files to move unless a directory is specified in the command line. This is the System V syntax of the install command and is what the RS/6000 uses by default.

AIX also provides the Berkeley install command, which has the syntax:

```
/usr/ucb/install [-c] [-m mode] [-o owner] [-g group] [-s] file dir
```

where -c copies the file to dir.
-m mode specifies the mode of the file (default 755).
-o owner specifies an owner other than your id.
-g group specifies a group other than your gid.
-s strips the file after installation.
file is file to move (or copy).
dir is destination directory.

Note that the syntax of the two commands is different. You may encounter problems with your installation scripts with some of the free software. The error messages will say something about being unable to move file dir, etc. This error message may be coming from the install command. Check the syntax of the makefile (or Makefile) as well as the syntax supported on your machine before you proceed.

Some very simple examples are:

```
$ install -c kevin /usr/bin (BSD style)
```

This will copy the file kevin into the /usr/bin directory. From then on you can execute kevin just as if it were a system-level command.

The command:

```
$ install -c /usr/bin kevin (SYSV style)
```

will accomplish the same as the previous Berkeley command. Note the difference in syntax and the problems that this may cause and beware.

The command:

```
$ install -i kevin /usr/local/bin, /usr/bin, /usr/kevin
```

will install a copy of kevin in /usr/local/bin, /usr/bin, and /usr/kevin if the file kevin exists. Remember that install only replaces existing files unless you use the -f dir option on the command. To force kevin into all three directories above, you might use:

```
$ install -f /usr/bin -o kevin
$ install -f /usr/bin -o kevin
$ install -f /usr/bin -o kevin
```

Note that you must execute this command three times to place it in three directories. Note also that the -o preserves any other kevin command in these directories and renames it OLDkevin.

There are a variety of ways you can use the install command. See your local documentation for more details.

4.9.3 Conclusion

install is a tool which allows you to place or replace files in directories from the command line. It provides a capability used by many makefiles to move files in and out of directories relatively transparently. Keep this in mind as you read through this book.

4.10 cb

4.10.1 Introduction

cb stands for C source beautifier and is a tool which formats C source files into more readable formats. It is a very powerful tool which assists the C developer in reading source code, particularly that written by other developers.

4.10.2 Usage

The basic syntax for cb is:

```
cb [-s][-l length | -j] [file ...]
```

where -s formats the output source code according to the K&R style.
-l length splits lines longer than length characters.
-j joins split lines.
file ... is one or more input files.

cb reads from standard input or specified files and directs to standard output, so you can use the standard shell manipulation characters. A simple example is:

```
$ cb test.c > newtest.c
```

This will generate the file newtest.c from the old file test.c.

4.10.3 Conclusion

cb is a tool which significantly enhances the readability of certain C source files. Through the use of indentation and formatting techniques, cb makes source files significantly more readable and understandable. It is particularly useful for reading other developers' source files.

4.11 cflow

4.11.1 Introduction

cflow is a tool which generates a flow graph of external references within a program or set of programs. You can use it to document all external references and calls between C source programs. This is particularly useful when beginning to examine other developers' code.

4.11.2 Usage

The sytax for cflow is:

```
cflow [-dnumber] [-Idir] [-i_] [-ip] [-ix] [-qDBCS] [-r] [-MA]
[-Uname] [-Ndnumber] [-Nlnumber] [-Nnnumber] [-Ntnumber]
[-Dname[=definition]] file ...
```

where -dnumber sets the depth of functions to the number to which the graph system goes.
-Idir adds dir to directory in which to search for #include files.
-i_ includes names that begin with an underline.
-ip disables ANSI function prototypes.
-ix includes static and external data symbols.
-qDBCS sets multibyte mode matching current locale.
-r produces an inverted listing.
-MA specifies that the first pass of the lint command is operated in ANSI mode. The default is extended mode.

-Uname removes the definition of the name parameter.
-Ndnumber changes the dimension table size to number. The default is 2000.
-Nlnumber changes the number of type nodes to number. The default is 8000.
-Nnnumber changes the symbol table size to number. The default is 1500.
-Ntnumber changes the number of tree nodes to number. The default is 1000.
-Dname=definition defines the name parameter; is similar to the #define statement.
file... specifies one or more files to analyze.

cflow works on C source, yacc, lex, assembler, and object files and writes the results of its analysis to standard output. It actually passes non-C source files such as yacc and lex through the compilation process, then analyzes the resulting C source code before generating its result. It also takes the symbols from the assembler files and produces its output.

A simple cflow example is:

```
$ cflow test1.c test2.c > test.output
```

This will produce a file test.output which contains the flow graphs of the test1.c and test2.c files. Another simple example is:

```
$ cflow scan.l
```

This will generate a flow graph of the lex intput file scan.l on the standard output. Remember that the file scan.l is run through lex before the cflow program analyzes the output.

4.11.3 Conclusion

The cflow command is very useful for understanding the relationship between C and other source files in a software system. Through the generation of flow graphs, you can quickly understand where dependencies and calls are created.

4.12 cxref

4.12.1 Introduction

cxref is a command which analyzes C source code files and produces a cross-reference table containing all symbols, including those in the #de-

fine statements. It is a very useful tool for analyzing and debugging unfamiliar C source code.

4.12.2 Usage

The syntax for cxref is:

```
cxref [-c] [-ofile] [-qDBCS] [-s] [-t] [-w [number]][-Dname
[=definition]] [-Idir] [-Uname] [-Ndnumber] [-Nlnumber] [-Nnnumber]
[-Ntnumber] file...
```

where -c displays the combined listing of cross-references of all input files.
-ofile specifies file as the output file.
-qDBCS specifies the multibyte character set.
-s does not display the input file names.
-t generates 80-column-wide listing.
-wnumber generates a listing number columns wide; number must be greater than 51.
-Dname=definition defines name as in a #define statement.
-Idir adds additional directories to search for #include files.
-Uname removes any definition of name.
-Ndnumber changes the dimension of table size to number. Default is 2000.
-Nlnumber changes the number of type nodes to number. Default is 8000.
-Nnnumber changes the symbol table size to number. The default is 1500.
-Ntnumber changes the number of tree nodes to number. The default is 1000.
file ... specifies one or more input filenames.

The only issue to watch out for is the function prototye issue. Function prototypes are handled in a special way. Old-style function declaration statments are displayed simply as the function prototype name without the optional prototype identifiers, whereas the newer ANSI-style function prototypes are fully listed, including optional prototype identifiers.

A simple example is:

```
$ cxref -c -t test1.c test2.c test3.c > test.cxref
```

This will generate an 80-column-wide combined cross-reference listing in test.cxref from the input files test1.c, test2.c and test3.c.

4.12.3 Conclusion

cxref is a very useful command for generating a functional listing of the separate source files making up a single software system. Use it in the initial stages of analyzing a software system to ensure that you have a complete understanding of how the program is functioning and where the dependencies are before proceeding to the change phase of the project.

4.13 tn3270

4.13.1 Introduction

tn3270 is an application which provides 3270 terminal emulation from a UNIX workstation. It requires no special hardware and can run with either the tn3270 protocol over a LAN or WAN. All that is required is that a transport such as TCP/IP be present on the connection. It is a tool which constantly amazes people, and after using it, they wonder why they have been buying expensive solutions that provide similar or less functionality.

There are several freeware versions of tn3270 available from the Internet; however, a reasonably good version of tn3270 comes with AIX and, therefore, this is the one discussed in this section.

4.13.2 Usage

The tn3270 protocol. The tn3270 protocol is a public domain protocol which runs above the transport layer of your network. It is approximately a layer 5 protocol in the seven-layer ISO model. It is a specification which describes full-screen 3270 data stream emulation on a non-3270 data stream device.

With the tn3270 protocol, you can distribute the 3270 data stream onto LANs and WANs that are running a TCP/IP transport. tn3270 uses the standard ports for telnet to provide this service and merely relies on the tn3270 emulation software package to be on the client end to interpret the contents of the delivered packets, break them apart, and transform them from 3270 to curses packets which UNIX uses to manipulate the screen.

Most implementation of TCP/IP for mainframe support the tn3270 protocol. Certainly IBM's TCP/IP for VM and MVS provide a tn3270 data stream with no additional configuration on the mainframe required. An example of the use of this architecture is shown in Fig. 4.1. Note that the tn3270 product does *not* provide any cluster controller emulation capabilites but instead looks like a 3278 dumb terminal device. There are other products which provide these sorts of capabilities,

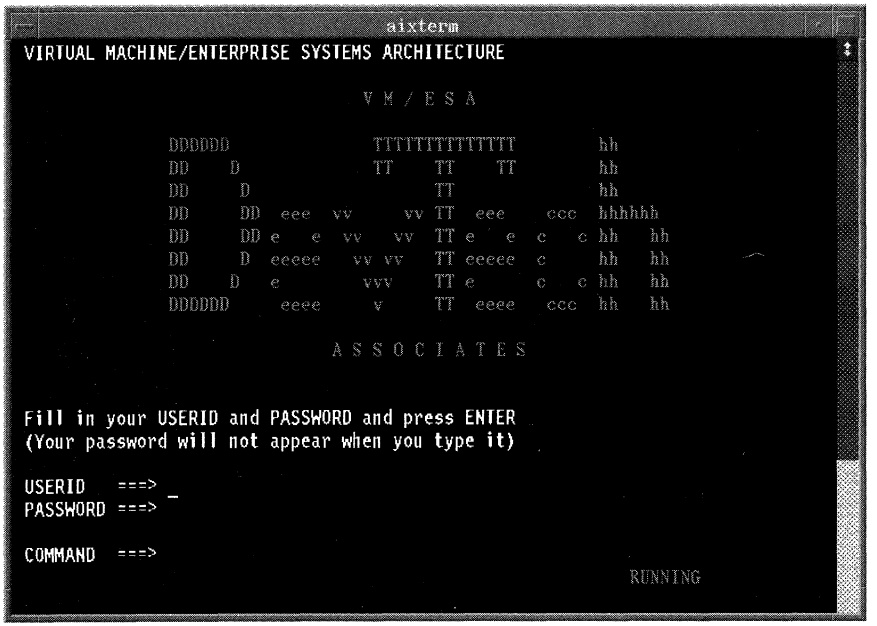

Figure 4.1 The tn3270 window.

and they are beyond the scope of this book. Suffice it to say that you can use various vendor products to provide local 317*x* and 327*x* cluster controller emulation capabilites, and you can use tn3270 to provide a dumb terminal emulation in these products.

To invoke tn3270, type the command:

```
$ tn3270 [-d] [-n filename] [-e termtype] [hostname [port]]
```

where -d turns on socket-level tracing.
-n filename is the file to receive tracing information; default is stderror.
-e termtype specifies a terminal type to emulate.
hostname is the hostname of the remote system.
port is used to specify a port other than the standard port.

When you invoke tn3270, a negotiation takes place between your terminal and the mainframe which establishes your terminal characteristics. The tn3270 application looks at your TERM variable and establishes things such as rows, columns, and keyboard mappings. In all cases the terminal looks like a 3278 to the mainframe, and the negotiation determines which model within the 3278 family is emulated.

tn3270 uses curses to map keyboard ASCII sequences to the signals

the mainframe is expecting. There is a default file which describes the mapping of all keys on the keyboard to keys and actions expected on the mainframe. /etc/map3270 contains example codes and mappings for the tn3270 product. This describes the default characteristics of the terminal emulation if you make no changes. You can also create your own mapping file named $HOME/.3270keys which contains information in a similar format to /etc/map3270 but allows you to define your own key mappings. In addition, tn3270 looks for an environmental variable MAP3270 to define the mapping file. Use this to point to your own defined key mapping files.

Modes of operation. There are two modes of operation with tn3270. One is command mode, which gives you a prompt tn3270> (the other is full-screen mode). From here you can issue tn3270 commands which control most aspects of your tn3270 session. A key command is the help. Type:

```
tn3270> help
```

Commands may be abbreviated. Commands are:

`close`	Closes current connection
`display`	Displays operating parameters
`emulate`	Emulates a vt100 or 3270 terminal
`mode`	Tries to enter line-by-line or character-at-a-time mode
`open`	Connects to a site
`quit`	Exits telnet
`send`	Transmits special characters ('send ?' for more)
`set`	Sets operating parameters ('set ?' for more)
`status`	Prints status information
`toggle`	Toggles operating parameters ('toggle ?' for more)
`z`	Suspends telnet
`?`	Prints help information

This is a listing of the commands available within the tn3270 command mode. Note that these are very similar to the standard telnet commands.

If you don't include a hostname on the tn3270 line, you will be placed in command mode. To open a connection to a remote machine using the 3270 data stream issue the command:

```
tn3270>open hostname
```

This will connect you to the remote host, negotiate a session and terminal characteristics, and put you at the login screen just as if you were

sitting on a 3270 data stream full-screen terminal. You are now in full-screen emulation mode. From this mode you can execute all commands you normally would from a full-screen 3278-style terminal. A mapping has occurred, as is discussed in the following section.

You can move from full-screen emulation mode to command mode by typing CTRL-C. This will take you to a tn3270> prompt where you can issue command mode commands as you normally would. To return to full-screen mode, type <return> on a blank line. While you are in command mode, the full-screen session is merely suspended. When you reenter the full-screen session, you should return to your previous full-screen state.

There are many commands within the command mode for tn3270. In fact, the full set for telnet is supported. You can use quit and open to quit and open new connections to other machines as well as use a variety of other commands from with tn3270 to control your environment. Use the interactive help for more information on which commands may be useful to you.

Terminal emulation issues. The best way to use the tn3270 tool is through X11. Simply move into a terminal shell window on your local machines and invoke the tn3270 exactly as described above. The emulation will be taken care of by your machine.

It gets more interesting, however, when you are accessing a tn3270 server remotely. In other words, the tn3270 application runs on a node other than your local one. There are two ways to use the tn3270 product effectively on a remote station. The first is to use the X11 capability and the xterm terminal emulator to provide remote support:

```
$ TERM=vt100;export TERM /* export the TERM variable as a standard
vt100 */
$ telnet remotehost /* go to the remote machine which is running
tn3270 */
login
$ /usr/lpp/X11/bin/xterm -display localhost:0
/*this pops up an xterm window on your local display...select this
window to make it active*/
$ tn3270
/*now you have invoked the tn3270 and used the X11 server to provide
terminal emulation*/
```

The second way is to set both the local TERM and remote TERM to vt100 and use the existing window:

```
$ TERM=vt100;export TERM
$ telnet remotehost
login
$ TERM=vt100;export TERM
$ tn3270
```

Note that both solutions work, but the first is more elegant in terms of full-screen terminal emulation and support for windowing functions. You could, and probably should, set up a rsh-type command to invoke the shell from a remote machine. For example:

```
$ rsh remotehost -l usename /usr/lpp/X11/bin/xterm
$ tn3270 /* from within the newly created window */
```

Keyboard mapping issues. As discussed earlier, when tn3270 is invoked, it looks first for a file $HOME/.3270keys and then for a file /etc/map3270. These files contain the default keyboard mappings for tn3270. There is a manual page on map3270 that gives more information on the exact structure of the keyboard mapping files. See this and below for more information.

To avoid having to scan the /etc/map3270 file every time tn3270 is invoked, you can set the environmental variable MAP3270, which is read before tn3270 scans the /etc/map3270 file. MAP3270 conains either a fully qualified path (beginning with a /) which points to a file which contains the keyboard mappings for your particular terminal or contains actual keyboard charactersitics and key mappings. The tn3270 tool scans the string contained in MAP3270. If it begins with a /, it looks for a file with a name matching the string which contains the mappings. If the MAP3270 variable does not begin with a /, the tn3270 uses the mappings contained within the MAP3270 variable itself to establish mappings for your session.

The /etc/map3270 file is a database file which contains a listing of terminal types and corresponding keyboard mappings. The final step in building tn3270 is to move a copy of this file to the /etc directory. Now let's take a look at the structure of the map3270 file. There are many terminal types represented in the map3270 database. The vt100 section looks as follows:

```
vt100 | vt100nam | pt100 | vt125 | vt102 | direct831 | tek4125 |
pcplot | microvax{ enter = '^m'; clear = '^z' | '\EOM';

nl = '^?'; tab = '^i'; btab = '^b'; left = '^h' | '\EOD'; right
= '^l' | '\EOC'; up = '^k' | '\EOA'; down = '^j' | '\EOB'; home =
'\EOn';

delete = '^d'; eeof = '^e'; einp = '^w'; insrt = '^ ' | '\E ';

# pf keys pfk1 = '\EOq' | '\E1'; pfk2 = '\EOr' | '\E2'; pfk3
= '\EOs' | '\E3'; pfk4 = '\EOt' | '\E4'; pfk5 = '\EOu' | '\E5'; pfk6
= '\EOv' | '\E6'; pfk7 = '\EOw' | '\E7'; pfk8 = '\EOx' | '\E8'; pfk9
= '\EOy' | '\E9'; pfk10 = '\EOP\EOp' | '\E0'; pfk11 = '\EOP\EOq' |
'\E-'; pfk12 = '\EOP\EOr' | '\E='; pfk13 = '\EOP\EOs' | '^f13';
pfk14 = '\EOP\EOt' | '^f14'; pfk15 = '\EOP\EOu' | '^f15'; pfk16
= '\EOP\EOv' | '^f16'; pfk17 = '\EOP\EOw' | '^f17'; pfk18
= '\EOP\EOx' | '^f18'; pfk19 = '\EOP\EOy' | '^f19'; pfk20
= '\EOQ\EOp' | '^f20'; pfk21 = '\EOQ\EOq' | '^f21';
```

```
# program attention keys pa1 = '\E\EOP' | '^p1'; pa2 = '\E\EOQ' |
'^p2';

# local control keys
escape = '^c';# escape to telnet command mode master_reset = '^g';
centsign = '^\';

# local editing keys settab = '\E;'; deltab
= '\E\''; clrtab = '\E:'; setmrg = '\E,'; sethom = '\E.'; coltab
= '\E\E[B'; colbak = '\E\E[A'; indent = '\E\E[C'; undent = '\E\E[D';
} # end of vt100,
etc. sun {
```

The first line describes all terminal types supported by the following definitions. The next fields describe characteristics of the ENTER key, CLEAR SCREEN key, newline (nl), TAB, and cursor movement keys. The sections entitled pf keys describes how the traditional PF keys are mapped. The vertical bar denotes options that perform the same task. The following characters are also special:

'/E'	ESC
'/n'	Newline
'/t'	TAB
'/r'	Carriage Return
^	CRTL

So, if you want to understand how to execute a PF1 as you would from a normal 3278-style terminal to access a help function, you would type:

```
ESCAPE 1
```

In general, the PF keys are mapped with an ESC and the associated numberic key across the top of the keyboard. Note that you should avoid using the keypad since this may be mapped to something else. To generate an interrupt (PA2), you would type:

```
Control-P 2
```

Again CRTL-C takes you back to command mode. You can create your own terminal definitions and map the keys appropriately and simply include it in this file. The map3270 manual page contains a very nice description of all functions supported and mapped by the map2370 database. See this for more information.The most commonly used keys are (remember ^ is CRTL):

^z	CLEAR
^p2	ATTN
^m	ENTER
^t	RESET

```
PF1             ESC 1
PF2             ESC 2
...etc...
PF13            ESC !
PF14            ESC @
...etc...
^u              ERASE
```

If you have problems with the terminal emulation and keyboard mappings, take a look at the termcap database for a listing of supported ASCII terminals. This should not be necessary since most ASCII terminals support the vt100 emulation, and this should provide reasonable emulation for you. However, if you have a keyboard which has special keys that you would like to use, consult the termcap and terminfo databases for more information.

4.13.3 Conclusion

This kind of tool will allow you to dial in from home with a dumb terminal or PC running ASCII full-screen emulation (vt100 or something similar) and log on to a full-screen 3270 environment. This same tool will allow you to access a full-screen 3270 environment from any workstation on any LAN that is connected to the 3270 environment.

Chapter

5

Native AIX Software Development Scripting Tools

5.1 Introduction

While Chap. 4 focused on native UNIX tools which provided functionality based on command sets and syntax, this chapter focuses on native tools which provide scripting and development capabilities well beyond those described in Chap. 4. The tools described in this chapter are very flexible and powerful ones to use to develop applications and tools on your own.

5.2 awk

5.2.1 Introduction

awk is an interactive programming language which provides significant function similar to a fourth-generation language in common business nomenclature. awk provides pattern recognition and manipulation capabilities. It is typically used to manipulate large pieces of text without actually having to modify or even edit the file. It is a very powerful tool and one that is certainly underutilized on most AIX computers. The name *awk* says much about AIX and the way in which it was developed. awk stands for the last names of each of the authors: Alfred Aho, Peter Weinberger, and Brian Kernighan. Modesty has never been a characteristic of most UNIX developers.

awk is really a programming language all by itself. It is one of the most powerful pattern recognition and manipulations languages available on AIX. It is a language which is C-like in syntax but is optimized to search files for strings and perform subsequent operations on these

input lines. It uses ed commands to search for regular expressions in strings within files and performs a specified action on them. While this chapter cannot begin to describe all the features and functions of awk, it does present the main areas of functionality and provides enough information to allow you to decide whether you should look into awk in more detail.

5.2.2 Usage

awk is invoked as follows:

```
awk [-Fx] -f program [file1 file2 ...]
```

where -Fx allows you to specify a separator x other than whitespace.
-f program specifies a file which contains the awk commands.
[file1 file2 ...] contains a list of input files separated by blanks.

awk contains many of the features you would find in third-generation languages such as conditional branching, looping, string and arithmetic variables, and output format statements. It also contains things that you don't see in most languages such as transparent typing of variables and very flexible syntax. This allows you to code very powerful awk programs without being concerned with variable typing, definition, and manipulation.

The awk program represented by the -f program in the awk command syntax contains the following general syntax:

```
pattern command {action}
```

where pattern command is an ed command which provides for string searching and manipulation, and the action part consists of C-like commands which perform actions on the output of the pattern command. awk performs all actions on all lines selected by the pattern part of the awk program. The best way to understand this is to take the example input file called data1:

```
1col1 1col2
2col1 2col2
3col1 3col2
4col1 4col2
```

This corresponds to the file part of the awk command. In other words, this will be the file that awk performs its program against. Let's also assume that we have a program called awk1 which looks like:

```
/3col1/ {print $1, $2}
```

To invoke this program against the above file, you would type:

```
$ awk -f awk1 data1
3col1 3col2
```

The output of that command is 3col1 3col2. awk first looked at the file awk1 for a pattern or ed command to use when examining the file data1. The /3col1/ command is the ed command to search for the string 3col1. awk performed the search and action pair on each line in the input file. In other words, it scanned line 1 and didn't find a match. It scanned line 2 and didn't find a match. It scanned line 3 and did find a match. It then applied the action print $1, $2 (more about this later) to this line and produced the output shown below the awk command. Finally, it scanned line 4 and didn't find a match. Once it reached the end of the data1 file, awk exited. This basic model holds true for all awk invocations; however, the awk program (awk1) can get much more powerful.

You can also execute the awk command functions from the command line by surrounding them with single quotes to prevent shell interpretation. For example, to execute the above awk commands without using the awk1 file, type:

```
$ awk '/3col1/ {print $1,$2}'
3col1 3col
```

If you place several files separated by blanks on the awk command line, awk will process one line at a time and step sequentially through each file on the command line. If you choose to use standard input, use a -. For example:

```
$ awk -f awk1 -
1col1 1col2
2col1 2col2
3col1 3col2
3col1 3col2
4col1 4col2
```

Note that as you type each line in, awk processes it and presents the results to standard out. In this case the awk1 program searched for 3col1 and thus matched the third line.

The awk language. The awk language is a very powerful one which consists of most functions you would expect within a procedural language. Patterns can be ed commands that match patterns within the data files and regular expressions as well. The regular expressions consist of objects and operators. awk statements can combine both string and arith-

metic operations in the same statement. Statements are terminated by NEWLINE or a semicolon. Also, just as in C, awk treats all statements within curly braces as a single statement. This allows you to nest statements under conditional branches just as you would with most other procedural languages.

Looping and conditional statements. The if conditional statement looks like:

```
if (condition) [{ statement }]
```

If you have a single statement, the curly braces are unnecessary. However, if you have more than a single statement, you must enclose them in curly braces to ensure that all are executed under the condition. For example:

```
{if (i<10) {
   print i
   ++i }
}
```

Note that the outside curly braces are necessary to denote that this is the action part of the awk statement while the inner braces are necessary to group the two statements together under the conditional statement. If you do not include a statement, the default action {print} is performed. This merely performs a print of the entire matched line.

There is also a while loop which looks like:

```
while(condition) [{] condition [}]
```

where the condition is similar to the if conditional, and the curly braces must be used to contain more than one statement if you want them treated as one statement group under the while command.

There is a do loop in nawk (see the nawk section below for more information on nawk). It looks like:

```
do [{] action [}] while (condition)
```

Finally, there is a for statement which looks like:

```
for (initcounter;test;increment) action
```

where initcounter sets the initial value for a loop counter.
test is the condition that is tested.
increment is the number to increment initcounter each time.

An example is in the examples section.

There are two other commands that affect command execution and flow:

`break`	Breaks completely out of a loop and begins at first line outside of loop
`continue`	Begins at top of loop in next iteration of loop itself

Besides string manipulation, numeric operations are fully suported by awk. The standard arithmetic statements supported are:

=	Assignment
<	Less than
>	Greater than
++	Increment by one
--	Decrement by one
/	Divide
*	Multiply
+	Add
-	Subtract
+=	Add expression following operator to variable preceding it
-=	Subtracts expression following operator to variable preceding it

All numbers are converted to floating point before being manipulated, which eases a lot of problems you normally experience when programming. There are also relational operators supported by awk which allow for comparisons within the condition section of the statement. The primary relational operators supported by awk are:

<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to
~	Matches
~!	Does not match

You can also separate multiple patterns with boolean operators such as:

&&	AND
||	OR
,	Range

See the examples for more discussion of the boolean operators.

Variables do not need to be initialized. This allows you to set the value of a variable without declaring it. For example:

```
s = 3
```

assigns the value of 3 to the variable s. It is not necessary to declare s anywhere else in the awk program. There are several special variables which are predefined:

`$0`	Current record
`$1-$n`	Fields in the current record
`FILENAME`	Name of current data file
`FS`	Input field separator (default space)
`NF`	Number of fields in current record
`NR`	Number of current record
`OFS`	Output field separator (default space)
`ORS`	Output record separator (default NEWLINE)
`RS`	Input record separator (default space)

Several of these can be changed either by invocation switches (-F) or within the awk program.

There are are also standard string functions which are supported by awk:

`index(string1,string2)`	Returns index of string2 in string1
`length[(string)]`	Returns the length of string; without an argument returns line length
`split(string,arr,del)`	Places elements of string, delimited by del in array arr[]
`sprintf(fmt,args)`	Defines formatted output args in format defined by fmt
`substr(string,pos,length)`	Returns string that begins at pos and is length characters long

All string functions act like their C language equivalents. See help on sprintf or other functions for more details about exact syntax and usage. See also some examples of function usage.

Finally, there are arithmetic operators in awk:

cos(x)

sin(x)

int(x)

log(x)

sqrt(x)

It is rare that you will need functions like these, but it is nice to know they are there.

BEGIN and END functions. You can structure special clauses, before and after the actual awk program, which are executed before and after the awk program reads the data input file. For example:

```
BEGIN {
      print "this is the beginning..."
      }
      {
      awk program commands...
      }
END   {
      print "all done now..."
      }
```

The BEGIN and END clauses can do computations based on numbers generated in the main body of the awk program. A classic example of using the END function is to calculate the sum and mean of a set of numbers in a column.

```
    {
    m += $1
    n++
    }
END {
    print "mean is",m/n, "number of items is",n
    }
```

Remember that the END is processed after all lines are read. This makes end perfect to perform numerical calculations on an entire set of numbers. You can use this much as you would a spreadsheet to perform calculations on columns of numbers.

Errors. awk is notorious for ignoring errors and simply producing garbage output. You have to be extremely careful when structuring and coding your awk program. This is why it is almost always recommended to create an awk command file instead of using the command line since this will allow you to change and iterate your program severals times to eliminate all possible errors. When you place the awk commands on the command line, you run the risk of the shell command interpreter doing something to them before routing them to the awk command. While there is nothing wrong with using the command line, experience dictates that the awk command file is a better way to go. In the interests of space, this book uses most examples on the command line; however, this is not an endorsement of this technique.

Passing parameters into a script. You can pass variables into awk programs by placing assignment statements between the script and data file name. For example:

```
$ awk -f awk1 var1=1 var2=2 data1
```

Note that the var1 and var2 statements must not contain spaces. Once you have invoked the awk1 script, the variables var1 and var2 are ac-

cessible to the script itself. For example, if you invoked awk1 as above and awk1 looked like:

```
{print var1, var2}
```

Your output would be:

```
1 2
1 2
1 2
1 2
```

Remember that each line is processed and the command executed.

Note that command line parameters are not available to the BEGIN section of the awk program. With nawk, there is a -v option that allows for command line parameters to be available to the BEGIN procedure through the ARGC and ARGV parameters (similar to argc and argv in C).

Arrays. In awk, all arrays are associative. This means that the array index can be a string or number. Arrays look like:

```
array[index] = value
```

where index represents a position within the array.
value assigns the value to array[index].

The structure to access and loop through this array structure is:

```
for (elem in array) action
```

where elem is the variable that takes on value of each array element.
array is the array name.
action is the action taken for each element in array.

Some examples. There are almost an infinite number of possible examples for awk. Below are some examples which illustrate some of its uses. Let's use the same data file as before (data1):

```
1col1 1col2
2col1 2col2
3col1 3col2
4col1 4col2
```

Example 1

```
$ awk '/1/' data1
1col1 1col2
```

This uses the default action of printing the entire line.

Example 2

```
$ awk '$1 ~/1/' data1
1col1 1col2
2col1 2col2
3col1 3col2
4col1 4col2
```

Example 3. Example 2 matches all 1s in the first column (~) and performs the default action (print). If you want to check for all matching strings which begin with a 1, use:

```
# awk '$1 ~/^1/' data1
1col1 1col2
```

Example 4

```
$ awk '$1 == 2col1' data1
2col1 2col2
```

This example uses a boolean operator to compare the first column ($!) with the string 2col1. Remember that you can use any regular expression.

Example 5. You can combine any number of functions and regular expressions to accomplish what you want:

```
$ awk 'length > 11 {print NR}' data1
```

This will scan the data1 file for lines longer than 11 characters. Note that the length function returns the length of the entire line including separators:

```
$ awk '{print length}' data1
11
11
11
11
```

Example 6. To print out the middle two lines, you could use:

```
$ awk 'NR == 2, NR == 3 {print}' data1
2col1 2col2
3col1 3col2
```

Note that the print command is redundant since this is the default action.

Example 7. You can also redefine variables and fields on the fly before output. For example, create an awk file called rename as shown below:

```
BEGIN {
      print "Changing stuff...here we go..."
}
{
if ($2 ~/2col2/) $2 = "2column2"
}
END   {
      print "Hope you're satisfied now..."
      }
```

Still operating on data1, you would see:

```
$ awk -f rename data1
2col1 2column2
```

Example 8. To see some examples of looping commands with flow control commands, use:

```
for (i=1;i<2;++i)
       if ($i == "2col2") {
       print i, $i
       break
       }
```

This example will loop through an input file looking at the first two fields until it finds a column match for 2col2; it then prints out the value of i and the column at i and breaks out of the loop.

Example 9. This example scans the passwd file for accounts without passwords and users with duplicate user id's. It demonstrates many of the features available in awk. (This example is taken from *A Practical Guide to the UNIX System,* by Mark G. Sobell, Benjamin/Cummings, 1989.)

```
awk < /etc/passwd ' BEGIN{
uid[void] = " " #tell awk that uid is an array
}
{      # no pattern indicates process all records
dup = 0
split($0,field,":") #split fields delimited by :
if (field[2] == "") #check for null password field
{
       if (field[5] == "") #check for null info field
       {
       print field[1] "has no password"
       }
else
       {
       print field[1] " (" field[5]") has no password"
       }
```

```
}
for (name in uid) == field[3] #loop through uid array
{
        if (uid[name] == field[3]) #check for 2nd use of id
        {
        print field[1] "has the same UID as "\
        name " : UID = " uid[name]
        dup = 1 #set duplicate flag
        }
}
if (!dup) #same as if (dup==0)
        {
        uid[field[1]] = field[3]
        }
}'
```

There are many things to note about the above file including comments, standard input redirection, and arrays. See awk help and books such as the O'Reilly and Associates Nutshell books for more informaton. Generate a sample file in the structure of a standard password file and try the above program out for yourself. Note that you invoke it by simply typing its name. This is a complete awk invocation in itself. For example, if the file is named checkpasswd, simply type:

```
$ checkpasswd
```

to run this awk program.

nawk. nawk stands for new awk and was released with SVR3. Many UNIX machines treat awk as nawk and don't tell you. nawk contains a richer set of commands and functions. Some of the newer functions are:

Multidimensional arrays

ARGV and ARGC system variables

Arithmetic functions such as atan2, rand, srand

String substitution commands such as sub and gsub

Writing your own functions

System access via the system() call

Again, it should be said that many of these functions have been integrated into awk as awk has been replaced by nawk. However, you will still find awk out there, and you should be careful how you code your awk scripts if portability is an issue. Perhaps GNU's awk (gawk) may be of interest if you are concerned about portability.

gawk. This is GNU's version of awk. It contains functions not in awk or nawk and runs on almost all platforms running today. See Sec. 7.11

in this book as well as the software included on the accompanying CD for more information about gawk.

awkcc. awkcc is a utility which converts awk programs to C programs which can then be compiled and executed. Because awk is an interpreted language, it is slow and relatively clumsy when it comes to execution and performance. The awkcc program is available from the AT&T System Toolchest. Call AT&T for more information.

5.2.3 awk changes in AIX 4.1

There are some minor changes to awk in AIX 4.1 which need to be mentioned here. They are:

1. First, the usage message will be issued and a nonzero value will be returned when invalid flags are specified.
2. awk now counts comment lines once instead of twice.
3. Variable assignments to the BEGIN procedure are no longer available from the command line. In AIX 4.1, they must be made using the -v variable=value option before the -f file commands option is specified on the command line. This is in conformance to XPG4 and POSIX specifications.

5.2.4 Conclusion

awk is a very powerful language for file and string manipulation. Its strengths occur when data is formatted in such a way that it can be manipulated by column. You can treat data as type independent and awk will behave as you would expect most of the time. Spend some time with awk and you will begin to see some of its power. If you are interested in the GNU version of awk, see Sec. 7.11. This has the advantage of being the same awk on all platforms in your environment, while awk can vary from UNIX to UNIX implementation.

5.3 sed

5.3.1 Introduction

sed is an acronym for stream editor. It interprets scripts written in sed format. It supports the basic functions of ed while having an interactive capability of grep. ed reads in one line at a time and performs a sed command against it, it then reads in the next line, and so on. If there is a match in the current line, the substitution is made and the resulting

line printed out. If there is no match, the current line is printed out unchanged.

sed is most often used to perform substitutions in medium- and large-size files. Because of the ability to act on one line at a time, you can alter very large files without invoking an editor or worrying about memory or disk space requirements. Many editors, including vi and ed, read a file from the disk into virtual memory and create a temporary or working file. This essentially doubles the disk space required and may cause problems when working with large files. sed helps you avoid this, and many AIX developers use sed for just this reason.

5.3.2 Usage

The syntax for the sed command is:

```
sed [-n] [-e sedcommand[-e sedcommand...]] [-f scriptfile] [filelist]
```

or:

```
sed "sed command(s)" [filelist]
```

where -n means no print; sed does not copy files to stdout except as specified by the p command.
-e sedcommand allows you to enter multiple sed commands on the command line without having to create a file.
-f scriptfile specifies a sed command script.
filelist is a list of files separated by blanks to be processed; if filelist is not specified, standard input (stdin) is used, which means the keyboard.

Simple, short sed commands are usually entered on the command line, while more complex and lengthy sed scripts are typically invoked from a file containing multiple sed commands.

The sed command uses standard ed commands and performs substitutions that you would normally expect to enter within an editor. For example, examine the file named file.text below:

```
john is great
pete is good
gerard is cranky
sam is bad
joe is good
frank is frank, what can you say
```

If you wanted to change a simple string within the above file, you could issue the command:

```
$ sed "s/john/kevin/" file.txt
```

The resultant output would be:

```
kevin is great
pete is good
gerard is cranky
sam is bad
joe is good
frank is frank, what can you say
```

The above command substitutes the first occurrence of jim in every line with kevin. To substitute for good, you would use:

```
$ sed "s/good/okay/" file.txt
```

The resulting output is:

```
kevin is great
pete is okay
gerard is cranky
sam is bad
joe is okay
frank is frank, what can you say
```

You can issue all of the commands in the ed editor such as deletes, copies, moves, and include lines numbers for ranges. For example:

```
$ sed "1,3s/is/is not/" file.txt

kevin is not great
pete is not okay
gerard is not cranky
sam is bad
joe is okay
frank is frank, what can you say
```

Finally, if you want to replace multiple instances on the same line, you would use the /g switch as follows:

```
sed "s/frank/kevin/g" file.txt

john is great
pete is good
gerard is cranky
sam is bad
joe is good
kevin is kevin, what can you say
```

If you had not included the /g option (for global), it would have only substituted kevin for the first occurrence of frank, and the subsequent line would have been:

```
kevin is frank, what can you say
```

Note that the sed command is included in double quotes to prevent the shell from interpreting the contents before passing it to the sed executable. This is a common requirement in UNIX since the shells like the C shell and Bourne shell parse and process all command line information before passing it to any commands or utilities. To ensure no shell preprocessing, simply include any information in double quotes.

sed scripts. sed command files typically consist of lines of the following format:

```
[address[,address]] instruction [arguments]
```

where address consists of line numbers (and special characters such as $ and ^) separated by commands to denote a range.
instruction is an editing instruction that modifies the text.
arguments are commands dependent on the instruction; see ed syntax for more information.

You can include all sed commands in a file and invoke the file from the command line to provide the sed commands. For example, the sed input file (named sed.input) might look like:

```
s/oldstring/newstring/
/newstring/d
```

The example data file called file.data looks like:

```
this contains oldstring
this doesn't contain oldstring
this contains oldstring
this doesn't contain oldstring
```

To invoke the sed script file on the above data file, you would type:

```
$ sed -f sed.input file.data
this doesn't contain oldstring
this doesn't contain oldstring
```

To understand what sed did in this context, you must understand how sed processes sed scripts and input data files. sed first reads in the first line of data.file and processes the entire sed script file against this line before moving to the second line of file.data. This means that sed reads in the first line:

```
this contains oldstring
```

and performs a string substitution resulting in the string:

```
this contains newstring
```

It then performs the next command in the sed script file, which deletes the line which contains the string newstring, which this line does. It deletes the line. sed has reached the end of the sed script file and reads the next line in the input file file.data. It then replicates the above procedure; however, because there is no string substitution, the line is not deleted. The line is printed out and sed moves to the next line in the intput file. This occurs for all lines in the input data file.

The above discussion may concern you; if it doesn't it should. It is often very difficult to predict exactly what is going to happen when you apply multiple edits to a file with sed. Because of this, sed has the additional safeguard of writing the resultant lines to standard output and not to the original input file. If you want to write the resultant output to a file, you simply redirect standard output as you would with any other command:

```
$ sed "s/oldstring/newstring/" file.txt > newfile.txt
```

This will generate a file called newfile.txt with the resultant output. It is a good idea, however, to first save a copy of the file you are modifying before performing a sed on it since this will ensure you get the results you are expecting. If you have a backup copy, you can always recover from a mistake, but if you don't. . . .

Basic sed commands. Some of the basic sed commands are:

`a` Appends one or more lines to the current line. Append has a special format:

```
[address] a\
text\
text\
text
```

where the address must consist of a single number or defaults to the entire file, and the insertion text must be continued with backslashes (see examples).

`#` This is a comment line and must occur on the the first line only; the comment can be continued onto the next line with a backslash.

`c` Changes selected lines and replaces them with new text.

`d` Deletes the current line. Note that this causes sed to read the next line in the input data file when it is done processing the line even if there are other sed commands in the sed script file.

`i` Insert is exactly the same as append except that it places the insertion text on a line before the current line instead of appending onto the current line.

`l` Lists nonprintable characters as their ASCII code equivalents.

n	Next reads the next input line from the input data file. It also writes out the current line.
p	Prints current line as is with no future changes caused by sed script commands.
q	Quits processing in sed.
r	Reads the contents of a specified file and appends to the current line.
s	Substitute works exactly as in ed and vi; see examples above and below.
t	Transforms a character in a given position to another (see transform section).
w	Writes output to a specified file.

Blank lines and spaces. A useful example is the following:

```
$ sed "/^$/d" file.txt
```

This will remove all blank lines in the file named file.txt. The ^ represents the beginning of the line, and the $ represents the end. The lack of address means that this command will act on all lines, and the d means delete any lines that match the pattern of a blank line.

Note that, just as in ed, blank spaces within substitute strings are honored. For example, to remove a leading blank on each line, you could use the command:

```
$ sed "s/ //" file.txt
```

This would remove the first blank on each line in the file file.txt.

More about addresses. It is important to note that, just as in ed, sed can use strings matches as addresses. For example, if you have the following input file named input.dat:

```
this is stuff before the .include macro
blah blah blah
.include
this is an include part of the file because
it is in the include macro section contained by a .include directive
and a ..
..
this is other stuff not related to the include macro section
blah blah blah...
```

you can print out the include macro section with the command:

```
$ sed -n "/^\.include/,/^\.\./p" input.dat
.include
this is an include part of the file because
it is in the include macro section contained by a .include directive
and a ..
..
```

Note that the first address is derived by the resultant match of the .include macro and the last address to close the range is derived by the match of... Note also that the .s must be backslashed (\), which escapes them from the sed interpreter and ensures that they are interpreted literally.

Append, change, and insert. The append, change, and insert commands all have a similar syntax:

```
append [line]a\
text\
text

change [line,line]c\
text\
text

insert [line]i\
text\
text
```

where the text to be appended, changed, or inserted ends in a line without a backslash. Note also that the line cannot be a range of lines but must be a single line for both append and insert, while the change command can accept a range of lines.

Insert inserts any text before the line is matched. Append appends text to the end of the line matched in the line statement. Finally, change outputs the text *once* and deletes all lines in the range specified in the command.

An example data file called file.data might look something like:

```
root:S9KIMi9QQ4f8U:0:1:Operator:/:/bin/csh
nobody:*:65534:65534::/:
daemon:*:1:1::/:
sys:*:2:2::/:/bin/csh
bin:*:3:3::/bin:
uucp:*:4:8::/var/spool/uucppublic:
news:*:6:6::/var/spool/news:/bin/csh
```

You can use the insert command to insert information before the lines to be processed. An example sed script (sample.sed) might look like:

```
1i\
# This is the Password File for this Machine
```

You would get:

```
$ sed -f sample.sed data.input

# This is the Password File for this Machine
root:S9KIMi9QQ4f8U:0:1:Operator:/:/bin/csh
```

```
nobody:*:65534:65534::/:
daemon:*:1:1::/:
sys:*:2:2::/:/bin/csh
bin:*:3:3::/bin:
uucp:*:4:8::/var/spool/uucppublic:
news:*:6:6::/var/spool/news:/bin/csh
```

Transform. Transform is structured as follows:

```
[address]y/abc/xyz/
```

where the replacement is made character by character without regard to any characters around it. In other words, the above sed command will replace all *a* characters with an *x* character regardless of their positions or the surrounding characters. Where this is particularly useful is in changing uppercase to lowercase and vice versa. For example:

```
/.*/y/abcdefghijklmnopqrstuvwxyz/ABCDEFGHIJKLMNOPQRSTUVWXYZ/
```

will transform all lowercase letters to uppercase in a file.

Reading and writing files with sed. You can use the w and r commands to include files and write intermediate files with sed. The read command can be used as follows:

```
[address]r file
```

where you can specify an address to include a file to be processed. For example, if you have the following input data file named data.file:

```
this is not line1
this is not line2
this is not line3
```

and an additional file named data.file2 which contains:

```
this is line4
this is line5
this is line6
```

you can combine the files and process them as follows:

```
$ sed '$r data.file2' data.file

this is not line1
this is not line2
this is not line3
this is line4
this is line5
this is line6
```

The write command allows for the creation of multiple files from a single input data file. For example, if you have a data input file containing the following information:

```
sun gerard
ibm kevin
sun betty
sun carol
ibm john
ibm jeff
```

and you wanted to create separate files containing information broken down as machine types and names, you could build a script like:

```
/^sun/w sun.out
/^ibm/w ibm.out
```

This would create two separate files called sun.out and ibm.out containing a list of Sun users and a list of IBM users.

5.3.3 sed changes in AIX 4.1

There are some minor changes to sed in AIX 4.1 which need to be mentioned here. They are primarily related to output of sed. The two major changes are:

1. [2 addr]l outputs nonprintable characters as one 3-digit octal number, except for \\, \a, \b, \f, \n, \r, \t, and \v, which are output with the corresponding escape sequence.
2. Long lines are folded or wrapped at 72 characters with each end-of-line designated by a $.

5.3.4 Conclusion

sed is an extremely powerful stream editor which allows you to perform quick and easy changes to a file without entering an editor. It can be applied within shell scripts and other environments to provide incredible functionality and power. This chapter touched briefly on most of the commands and techniques you would use with sed, but there is much more capability which has not been explored. Techniques such as pattern and hold space are key to utilizing the more powerful aspects of sed. Finally, scripts can contain branching and conditional information which can provide for more functionality inside the sed script files. See both the sed documentation and the books in Sec. D.1 in the back of this book for more detail.

5.4 make

5.4.1 Introduction

The make utility is one of the most powerful and complex of all the tools in the AIX environment. Because of its tremendous complexity, this chapter will not address all of the functionality of make but instead will attempt to introduce you to the essential concepts and power of the make utility so that you can understand, modify, and use the make facility to build all Internet software products described in this book, as well as products of your own.

What you will find is that once you understand make and all it can do, you will use it for all of your software development. make is most useful in large projects which involve a lot of source code files and a somewhat complex compilation and linking environment.

5.4.2 Usage

make allows you to define a set of dependencies between files and object code and executables which defines the compilation process. make will ensure the minimum compilation necessary to rebuild a product which reflects all changes since the last rebuild. make can call any source compilers including Fortran, C, and COBOL and can rebuild the associated products. make can also be used with noncompiled code such as documentation text to maintain dependencies between files.

Let's assume you have a program which consists of 100 source code files. Ninety of those files are contained an an archive library. You also need to link in four other libraries from a different product. You may find a bug or want to enhance some functionality of the product. This may require changes to several source code files. When you have made those changes, you will need to recompile and relink your executable. Because of the large number of files and libraries, it would take quite a while to recompile, rearchive, and relink all source files into a new executable. What you would like to do is recompile only those files that were affected by your changes. make allows you to do this by defining all relationships and dependencies between all files and libraries at the beginning of the project in a makefile. Once this is done, you can simply invoke make after making changes, and the make facility will only recompile and relink only those files which were changed.

make requires an investment of time at the front of the development process in terms of building the makefile and testing it. However, once it is finished, you will save a tremendous amount of time when making changes since make will handle all rebuilds for you automatically. This is the power of make and why most AIX software developers use it.

The syntax for make is:

```
make [-d] [-e] [-i] [-k] [-n] [-p] [-q] [-r] [-S] [-s] [-t]
[-f file] [target ...] [macro ...]
```

where -d is debug mode.
-e overrides assignments with environmental variables.
-i ignores errors and continues processing.
-k stops processing current target if error occurs but continues with other unrelated targets.
-n displays commands but doesn't execute them. Useful when debugging.
-p displays set of default macros and dependencies.
-q checks to see if current executable is up to date.
-r doesn't use default rules.
-S terminates if any command receives an error.
-s doesn't display commands as they are executed. Silent mode.
-t touches files without recompiling anything. Used to fool make.
-f file is used to use a makefile other than the default.
- target ... specifies a target or targets contained in the makefile to execute.
macro ... You can specify one or more macros.

The makefile. When make is invoked, it looks for a file in the current directory unless otherwise directed. This file is typically named makefile or Makefile and is searched for in that order. Both are equivalent and define dependencies and actions which control make's behavior.

The typical makefile contains targets, associated files, and actions. The basic structure is:

```
target [target] ... :[:] [file] ... [`commands`] ... [;command]
[(tab) command]
```

where the first line is known as the dependency line. The dependency line contains a target which is used by make to direct execution, a colon, and a list of files and commands on which the target is dependent. Note the commands are contained within backquotes, and in UNIX and in make, this means that make will treat the output of the command enclosed in backquotes as the input to the current context. All commands on the lines following the dependency line contain commands which will be executed to build the target. All command lines *must* be preceded with a tab. Blanks are *not* a substitute for a tab and will cause make to fail. This is one of its many idiosyncracies.

The second colon represents a target that is used more than once. In other words, if you have a target that is defined more than once in a makefile, you must follow each target occurrence with two colons to in-

form make that you are using the target more than once. The format for each occurrence of target is the same as for the single occurrence.

Comments are preceded with a pound (#) sign and continue until the end of a line. You must precede each command line with a #, or make will interpret the line. Blank lines can also be used and will be ignored by make except in a command list under a dependency.

Commands are executed each in its own shell, and therefore communication between commands is not possible. As is standard with UNIX, each command gets its own process context and variable space, and this will not be maintained even within a command list under a dependency. Keep this in mind when you are building makefiles and are thinking about dependencies between commands under a dependency line. You can execute more than one command in a single shell by placing a \ at the end of a command line. This continues the command onto the next line. You can place multiple commands separated by \s, and they will run in a single shell. Therefore communication between them will be possible through the use of variables and other process contextsentivite information.

There are three special characters which control the behavior of commands within the makefile:

+	Executes this command even if the -n, -q, or -t commands are specified
-	Ignores errors returned from this command line
@	Does not print out this command when executing

These, in conjunction with command line switches, control the behavior and output of make and the makefile.

Commands can be continued onto the next line with a \ placed at the end of a line.

A simple makefile. An example of a simple makefile is as follows:

```
kevin.o: kevin.c
cc -c kevin.c
```

This tells make that there is a dependency between kevin.o and kevin.c, and to generate the target kevin.o from the input file kevin.c, make needs to issue the command cc -c to compile the C source file and generate an object output file.

To execute this file named makefile, you would simply type:

```
$ make
```

in the directory which contained the makefile. This example happens to be a default rule (see the section on default rules) and is unneces-

sary; however, it is a good example of a simple makefile and how to use make. Note that the tab before the cc command is essential since the make command will not work without this exact structure. For more complex makefiles, see the examples section.

Environment. Environmental variables play a key role in determining how make behaves. By altering these variables, you can change the characteristics of make and its behavior. You need to understand which variables to use and how to use them.

When you run make, it reads the environment and treates all variables as macro definitions. The order of processing of those macro definitions is as follows:

1. Make's own default rules
2. Environmental variables
3. Description files
4. Command line

Note that the last definition of a macro overrides the previous. This means that the command line macro definitions will override all previous definitions and that description file definitions will override environmental variables, etc.

The behavior of make is determined as much by command line switches as by definitions in files. You can use the MAKEFLAGS or MFLAGS variables and these will be examined, in that order, for make switches. For example:

```
$ export MAKEFLAGS=-n
$ make
```

will display commands but not run them. Remember that, because of the definition order described above, if there is a conflicting switch in the definition makefile, the -n will be overriden. You can also override this variable with a command line option.

When commands are invoked, a default shell is used. make first looks for the environmental variable MAKESHELL and then for SHELL. If neither are defined, the Bourne shell (/bin/sh) is used. To change the shell to Korn shell, for example, you would type:

```
$export MAKESHELL=/bin/ksh
```

and all subsequent commands executed within the context of the make procedure would be executed in the Korn shell. The same is true for the C shell.

The environment is also important relative to your current file searches and contexts. If you issue commands which rely on the existence of unqualified filenames, you will need to be concerned with your current working directory and the existence and accessibility of your files.

Default (internal) rules. make has default rules and dependencies which govern the behavior of make when no explicit dependencies and rules are defined in the makefile. You can use the -p command to view some of these rules:

```
$ make -p

Macros:
ALL = $(WERMIT)
MANEXT = 1
MANDIR = /usr/man/manl
BINDIR = /usr/local/bin
DESTDIR =
WERMIT = makewhat
CC2 = cc
SHAREDLIB =
EXT = o
BOOTFILE = /edition7
makewhat:
commands:
  @echo 'make what? You must tell which system to make C-Kermit for.'
  @echo Examples: make bsd43, make sys5, make sunos41, etc.
  @echo Please read the comments at the beginning of the makefile.

.C~.a:
commands:
       $(GET) $(GFLAGS) -p $< > $*.C
       $(CCC) -c $(CCFLAGS) $*.C
       $(AR) $(ARFLAGS) $@ $*.o
       rm -f $*.[Co]

.C.a:
commands:
       $(CCC) -c $(CCFLAGS) $<
       $(AR) $(ARFLAGS) $@ $*.o
       rm -f $*.o

.C.o:
commands:
       $(CCC) $(CCFLAGS) -c $<

.h~.h:
commands:
       $(GET) $(GFLAGS) -p $< > $*.h

.f~.a:
commands:
       $(GET) $(GFLAGS) -p $< > $*.f
       $(FC) -c $(FFLAGS) $*.f
       $(AR) $(ARFLAGS) $@ $*.o
       rm -f $*.[fo]
```

```
.f.a:
commands:
        $(FC) -c $(FFLAGS) $<
        $(AR) $(ARFLAGS) $@ $*.o
        rm -f $*.o

.s~.a:
commands:
        $(GET) $(GFLAGS) -p $< > $*.s
        $(AS) $(ASFLAGS) -o $*.o $*.s
        $(AR) $(ARFLAGS) $@ $*.o
        -rm -f $*.[so]

.c~.a:
commands:
        $(GET) $(GFLAGS) -p $< > $*.c
        $(CC) -c $(CFLAGS) $*.c
        $(AR) $(ARFLAGS) $@ $*.o
        rm -f $*.[co]

.c.a:
commands:
        $(CC) -c $(CFLAGS) $<
        $(AR) $(ARFLAGS) $@ $*.o
        rm -f $*.o

.l.c:
commands:
        $(LEX) $<
        mv lex.yy.c $@

.y~.c:
commands:
        $(GET) $(GFLAGS) -p $< > $*.y
        $(YACC) $(YFLAGS) $*.y
        mv y.tab.c $*.c
        -rm -f $*.y

.l~.o:
commands:
        $(GET) $(GFLAGS) -p $< > $*.l
        $(LEX) $(LFLAGS) $*.l
        $(CC) $(CFLAGS) -c lex.yy.c
        rm -f lex.yy.c $*.l
        mv lex.yy.o $*.o

.C~:
commands:
        $(GET) $(GFLAGS) -p $< > $*.C
        $(CCC) $(CCFLAGS) $(LDFLAGS) $*.C -o $*
        -rm -f $*.C

.C:
commands:
        $(CCC) $(CCFLAGS) $(LDFLAGS) $< -o $@
.h~
.h

.sh~:
commands:
        $(GET) $(GFLAGS) -p $< > $*.sh
        cp $*.sh $*; chmod 0777 $@
        -rm -f $*.sh
```

```
.sh:
commands:
        cp $< $@; chmod 0777 $@

.f:
commands:
        $(FC) $(FFLAGS) $(LDFLAGS) $< -o $@

.c~:
commands:
        $(GET) $(GFLAGS) -p $< > $*.c
        $(CC) $(CFLAGS) $(LDFLAGS) $*.c -o $*
        -rm -f $*.c

.c:
commands:
        $(CC) $(CFLAGS) $(LDFLAGS) $< -o $@

.o

.SUFFIXES:
depends on: .o .c .c~ .f .f~ .y .y~ .l .l~ .s .s~ .sh .sh~ .h .h~ .C
.C~
make what? You must tell which system to make C-Kermit for.
Examples: make bsd43, make sys5, make sunos41, etc.
Please read the comments at the beginning of the makefile.
```

The above is only a partial listing of the output of the make -p command but is representative of the kind of default or built-in rules make contains. All definitions for compilers and library formats have a default format for make which determines how things will be compiled and linked. Keep in mind that this differs from system to system. It is generally a good idea to do a make -p before invoking make on your makefile to understand any default rules that may affect your desired result.

This describes the default dependencies and rules defined in make without any explicit definitions. Keep these in mind when you are building your makefile.

You can disable the internal rules of make with the -r command. Each machine typically stores internal rules in a file which you can examine. On AIX this file is /usr/ccs/lib/make.cfg.

Suffixes and dependencies. Suffix rules consist of default definitions and predetermined behaviors based on file suffixes. For example, the .c suffix tells make that this is a C source input file, and the default action to produce a .o file is the cc -c command. This default rule tells you why the above simple makefile is unneccesary since this rule is already predefined. You could have built the makefile as:

```
kevin.o: kevin.c
```

without a command, and make would have known what to do.

You can modify or include suffix rules in the default list of suffix rules by including a .SUFFIXES section in your makefile. For example:

```
.SUFFIXES: .o .kev
```

defines an additional suffix rule that says that to create a .o file you can look, in addition to the default fules, for a file ending in .kev. Note that later you must define what you want this rule to entail by defining commands to execute for this rule.

The default suffix rules are:

```
.SUFFIXES: .o .c .c\~ .f .f\~ .y .y\~ .l .l\~ .s .s\~ .sh .sh\~ .h
.h\~ .a
```

This defines the order in which files are searched to produce a nonexplicitly defined behavior for a given file suffix. The \~ on the end of some of the suffixes allows for the use of Source Code Control System (SCCS), which is the standard way to maintain and control versions of software in AIX. The rest of the suffixes pertain to:

.o	Object code
.c	C source code
.c\~	C source code in SCCS
.f	Fortran source code
.f\~	Fortran source code in SCCS
.y	yacc code
.y\~	yacc code in SCCS
.l	lex code
.l\~	lex code in SCCS
.s	Assembler
.s\~	Assembler in SCCS
.sh	Shell source
.sh\~	Shell in SCCS
.h	Header file source
.h\~	Header file source in SCCS
.a	Archive library

If you want to clear the default suffix list, simply use the .SUFFIXES command with no suffixes. For example:

```
.SUFFIXES:
```

will clear all default suffix rules.

The order of the SUFFIXES rules is important since make interprets the SUFFIXES lines from left to right. This means that make will first try to find an associated C source input file before a Fortran source file

before a yacc, etc. Be careful when placing files with the same name in the directory in which make is going to build a product. Always remember the order of the SUFFIXES default list.

Suffix rules which are explicitly contained in your makefile override default rules. For example, if you define a .c.o suffix rule and an associated set of commands, this will override the default rule. For example:

```
.c.o:
      f77 $<
```

tells make to invoke the Fortran compiler to translate a .c file into a .o file. You should get a lot of syntax errors if you try this one. Obviously this doesn't make sense; however, you can do this and override the default cc invocation if you would like.

make is extremely flexible and allows you to define any relationship you would like based on your needs. Don't forget that make maintains various default rules and relationships, and these must be considered when defining your own.

Targets. Targets define how make will behave. You can specify a target on the command line which will force make to begin execution of the makefile at that target line, skipping any previous commands and targets. This is used to give the makefile different behaviors based on which target you choose. For example, it is common in Internet software to provide a target called clean. This will delete all object files and executables from the current product. This ensures that when you issue a new build command (make) you get completely new executables independent of the relationship between the source code dates and the object and executable dates. This is necessary when recompiling a product on an architecture that is different from the one the executables were originally built on.

There are default targets which perform special functions. Some of these are:

.DEFAULT	Commands following this target tell make what to do if it can find no commands or rules to generate a specific file or files.
.IGNORE	Tells make to ignore errors in the commands and continue processing.
.POSIX	Processes the makefile as the POSIX standard specifies.
.PRECIOUS [file ...]	Files named on this line are not removed if make is interrupted; if no file is specified, all files are the default.
.SILENT	Make does not display any commands during the make.
.SUFFIXES	Adds suffixes to the suffix list (see above section).

If you do not specify a target on the make command line, the first target in the makefile will be executed. Note that a target can call make

recursively to build other dependent products. Note also that a target can use other targets to build products, and in fact, this is often the case when building more sophisticated makefiles for large products.

Targets can appear more than once in a makefile; however, only the first occurrence of the target can contain a command section. The dependency list between the two can and should be different, but the commands to operate on the target will be the same. Remember that make processes the first target it sees if not given an explicit target and will therefore find the first target and then the second. If you need to use more than one set of commands for the same target, you must use the double colon option described in the makefile section of this chapter.

Macros. Macros are like traditional variables. The basic syntax for a macro definiton is:

```
oldstring = newstring
```

This defines oldstring to be replaced by newstring everytime it occurs. To designate what you want replaced, use the syntax:

```
$(oldstring)
```

Everytime the $(oldstring) occurs in the file, it is replaced by newstring. This allows you to code your makefile with a symbol and create one single macro definition at the beginning of the makefile. Using this, you can change one line in your makefile and the change is reflected throughout the entire makefile.

make has a set of default macros that consist of:

`$*`	Filename without the suffix of the input file
`$@`	Full target name of the current target
`$<`	Source files of an out-of-date module
`$$`	Represents an actual dollar sign
`$$@`	Represents the current target name
`$%`	Name of an archive library member
`$?`	List of out-of-date files; used with explicit files
`CC`	Default C compiler on the system
`AS`	Default assembler on the system
`CFLAGS`	Default flags for C compiler

These default macros are used constantly throughout most makefiles since they make it much easier to define a set of files and associated behaviors. The most commonly changed macros are the final three. While you can change their definition in the beginning of the makefile

with a definition statement like those defined above, you can also change the macro definition from the command line. For example:

```
$ make "CC=gcc"
```

will cause make to invoke the GNU C compiler instead of the standard C compiler which comes with the system. The other most commonly used macro is CFLAGS. You can use this on the command line to change the compiler flags used by make when building object codes. For example, you may want to run the debugger on the resultant executable. For this you need to use the -g switch on the cc command:

```
$ make "CFLAGS=-g"
```

This will enable debugging for the resultant executable.

Macros are most often used much as you would use an environmental variable in a shell. By creating a macro at the beginning of the makefile, you can issue a change to the value of the macro once, and it is reflected through the entire makefile instantly. An example is:

```
KEVDIR = /usr/kevin
INCDIR = $(KEVDIR)/include
LIBDIR = $(KEVDIR)/lib
CFLAGS = -O -c -g
kevin: kevin.o
       $(CC) $(CFLAGS) -o kevin kevin.o
kevin.o: kevin.c $(LIBDIR)/kevin.lib $(INCDIR)/kevin.inc
```

This will use the default rules to compile kevin.o and will use the explicit rules stated as macros to link kevin. This is a strange example; can you find anything wrong with it?

Library archives and related issues. Archives are groups of executable files placed together in a library for ease of linking and maintenance. Most archives use the suffix .a to signify their type. The command to use with archives is ar (see Sec. 4.5 for more information). Archives are a special entity to make since they are used so often when developing software in an AIX environment. Because of the history of archives, they are well understood, and therefore, there are default rules and behavior within make related to archives.

make has several default rules which apply to archive libraries:

`.c.a`	C source code to an archive library (described below)
`.c\~.a`	SCCS C source code to archive library
`.s\~.a`	SCCS assembler source to archive library
`.f.a`	Fortran source to archive library
`.f\~.a`	SCCS Fortran source to archive library

These rules apply to give you default behavior for taking both SCCS and normal source code and placing compiled objects into a library. This saves you from having to write the commands section to compile and archive the resultant object code.

If a target contains parentheses, make assumes you are using an archive. The string within the parentheses denotes a member within the achive. The basic structure is:

```
lib(kevin.o)
```

This denotes an archive library named lib and a member within this library named kevin.o. You typically want to build entire archives consisting of compiled subroutines. There is a default rule for the construction of archive libraries. It is:

```
.c.a:
        $(CC) -c $(CFLAGS) $<
              ar rv $@ $*.o
              rm -f $*.o
```

This rule breaks down as follows:

`.c.a`	Target, which denotes that the translation from a .c file to a .a file is determined by the commands that follow the target.
`$(CC) -c $(CFLAGS) $<`	The net result of the $(CC) command is to recompile any C source code files that have a more recent modification date than their associated member in the archive where: $(CC)—the CC macro has been previously defined and is, by default, the cc command. $(CFLAGS)—the CFLAGS macro has been predefined and contains C compiler switches. $< says that you should use any .c file with the same name as a module in the archive with the same filename.
`ar rv $@ $*.o`	Uses the archive (ar) command to replace all object modules with a more current .c source code file. This is done one member at a time.
`$@`	Denotes the target in process.
`$*.o`	Denotes the current .o file in process. rm -f $*.o

This default rule for archives describes a significant amount of the power of make and how you can use it to control your software development process. Typically, the way you would see the dependencies and targets structures for an archive is as follows:

```
library: library(file1.o) library(file2.o) library(file3.o)
library(file4.o)
```

The target is library, the archive name is library, and the members within the library are named file1.o through file4.o. If you run make on

a makefile which contains the line above, the default rule described earlier will be invoked, and make will examine the current directory for an archive named library.a. It will then examine file1.o's modification date and compare it to file1.c in the current directory. If file1.c's modification date is newer than file1.o in the archive library, make will invoke the C compiler, archive (with the replace option) the new module, and remove the resultant file1.o from the compilation. It will then do the same for file2.o through file4.o sequentially. This is a very powerful capability and allows you to save a tremendous amount of time when maintaining large software libraries and packages.

Include files. Include files are the most commonly changed and used aspect of a makefile. In most programming languages you include files which contain header and variable information that is common to more than one source code file. Dependencies between source code files and header/include files should also be described in the makefile. For example, let's assume we have a program which consists of four source code files named file1.f through file4.f and two header files named file1.h and file2.h. You should describe the dependencies between these files in the makefile. For example:

```
files: file1.o file2.o file3.o file4.o
# create file executable files
        f77 file1.o file2.o file3.o file4.o -o files
# now build targets referenced above
file1.o: file1.f
        f77 -c file1.f file1.h
file2.o: file2.f
        f77 -c file2.f file1.h
file3.o: file3.f
        f77 -c file3.f file2.h
file4.o: file4.f
        f77 -c file4.f file2.h
```

Note that in the above example, the first two object files (file1.o and file2.o) depend not only on their respective source code files but on an include file (file1.h) as well. You have told the make facility that this dependency exists by including the name of the include file in the dependency statement. By doing this, make will automatically build only those executables that are affected when an include file is changed.

For example, if you change file2.h and reissue the make command, only file3.f and file4.f will be recompiled. After these have been recompiled, files will be relinked with the original file1.o and file2.o and with the new file3.o and file4.o. While this is a fairly simple example, you can see how very complex systems can be described with the makefile and how make can save you a significant amount of time in rebuilding a product.

Examples. You have seen most of the techniques used by makefile developers in structuring the makefile. Once you understand these basic techniques, the biggest part of constructing a makefile is the tedious work of understanding all of the relationships and workings of your development environment. Now that you have seen some simple examples, the best way to understand make is to see a relatively complex example. Once you study this example, much of the above information will begin to make sense as a whole.

This example is taken directly from the AIX manual page shipped with AIX on an RS/6000. It is the makefile used to maintain make itself.

```
# Description file for the Make program
# Macro def: send to be printed
P = qprt
# Macro def: source filenames used
FILES = Makefile version.c defs main.c \
        doname.c misc.c files.c \
        dosy.c gram.y lex.c gcos.c
# Macro def: object filenames used
OBJECTS = versio .o main.o doname.o \
        misc.o files.o dosys.o \
        gram.o
# Macro def: lint program and flags
LINT = lint -p
# Macro def: C compiler flags
CFLAGS = -O
# make depends on the files specified
# in the OBJECTS macro definition
make: $(OBJECTS)
# Build make with the cc program
        cc $(CFLAGS) $(OBJECTS) -o make
# Show the file sizes
        @size make
# The object files depend on a file
# named defs
$(OBJECTS): defs
# The file gram.o depends on lex.c
# uses internal rules to build gram.o
gram.o: lex.c
clean:
        rm *.o gram.c
        -du
# Copy the newly created program
# to /usr/bin and delete the program
# from the current directory
install:
        @size make /usr/bin/make
        cp make /usr/bin/make; rm make
# Empty file ":print" depends on the
# files included in the macro FILES
print: #(FILES)
# Print the recently changed files
        pr $? | $P
# Change the date on the empty file,
# print, to show the date of the last
# printing
```

```
        touch print
# Check the date of the old
# file against the date
# of the newly created file
test:
        make -dp | grep -v TIME > 1zap
        /usr/bin/make -dp | grep -v TIME > 2zap
        diff 1zap 2zap
        rm 1zap 2zap
# The program, lint, depends on the
# files that are listed
lint:   dosys.c doname.c files.c main.c misc.c \
        version.c gram.c
# Run lint on the files listed
# LINT is an internal macro
        $(LINT) dosys.c doname.c files.c main.c \
        misc.c version.c gram.c
        rm gram.c
# Archive the files that build make
arch:
        ar uv /sys/source/s2/make.a $(FILES)
```

There are several things to note about this makefile. First is the macro definitions. There are spaces surrounding the = sign, but this is not required. Unlike most other AIX utilities, you can either put spaces around the = signs or not. Several macros are used to define lists of files and commands. FILES and OBJECTS define lists of files to be used in a later command, while LINT and P define commands themselves. Remember macros are simply strings which will be substituted before execution of the line in the makefile, so they can be anything you would like.

The extensive use of comments is excellent practice since makefiles tend to end up being maintained by someone other than the original author. Documenting the makefile is just as important as documenting the code itself. Note that each commands is executed in its own shell, one per command line, except for the case of the command "cp make /usr/bin/make ; rm make." Two commands will be executed in the same shell because of the use of a semicolon. This is a powerful technique you can employ if you want to execute more than one command in a single shell.

As the make begins, the first target is make. When make reaches this target, it begins to process the makefile. The objects listed in the OBJECTS macro are substituted on the line, and dependencies are built. The OBJECTS files are dependent on defs, which is another target which make finds. grame.o depends on a file lex.c and make will use default rules to build gram.o from lex.c. Note that this is a special case, and make has internal rules for yacc and lex files. In this case, gram.c is dependent on gram.y, which is a yacc file. First make invokes the yacc compiler on gram.y to create gram.c Then make invokes the C compiler on gram.c to create gram.o. This chapter will not detail this;

see make, yacc, and lex documentation for more details. Finally, all dependencies are established, and the make executable is built.

The output of the make command would be something like:

```
$ make

cc -O -c version.c
cc -O -c main.c
cc -O -c doname.c
cc -O -c misc.c
cc -O -c files.c
cc -O -c dosys.c
yacc gram.y
mv y.tab.c gram.c
cc -O -c gram.c
cc version.o main.o doname.o misc.o files.o dosys.o
gram.o -o make
13188+3348+3044 = 19580b = 046174b
```

Note that targets like print, test, and lint are not executed. To run lint on all files to check for syntax errors, you would use:

```
$ make lint
```

This would run lint on all files and would generate the proper output for syntax checking. Once the first target (in this case make) is done, execution is stopped. Changed files can be printed by issuing the command:

```
$ make print
```

This command outlines a sophisticated makefile, and if you understand all that occurred with this makefile, you are ready to use make for your project; if you aren't comfortable with this example, see your make man page and a book such as the O'Reilly & Associates' *Managing Projects with Make* by Oram and Talbot.

5.4.3 Conclusion

make is a very powerful tool for building and maintaining software systems on a variety of platforms, including AIX. It is used by virtually every tool on AIX that consists of more than just a few source files to be built. By using make, you significantly reduce the maintenance load on your developers while at the same time increasing the quality of your code. It is certainly a tool you will read more about as you go through this book.

5.5 lex

5.5.1 Introduction

lex stands for lexical analysis program generator. It uses its own language and syntax to generate programs called scanners. These programs read data and parse according to the rules structured in the lex input file. These are typically used as parsers and as scanner programs for input data. You can generate a relatively simple lex input program which will scan input for regular expression patterns and generate an action coded in C. This gives you a relatively simple way to structure code for input structures while at the same time having the power of the C language to execute an action based on some string or expression. This is the real power of lex and why many sophisticated AIX users use it.

From the lex input file, a C source code file is created named lex.yy.c. This file is known as a scanner and is compiled with a special lex library which contains functions which lex uses. The resultant executable is what you use to scan input and generate the appropriate actions.

There is a GNU program known as flex which is used by most AIX power users to replace lex. See Sec. 7.6 for more details on the lex syntax itself since flex is backward compatible with respect to most lex functionality.

Lexical analysis is the process of taking information and dividing into units typically called tokens. This process requires a great deal of complexity in terms of analysis and coding to provide the "tokenization" of text. This is what lex excels at.

lex is often used with an associated product called yacc, which is a parsing language. yacc is described more fully in the next section. lex and yacc work togther to provide a significant fourth-generation/RAD capability which is not found in many other systems. In fact, many AIX tools are written with lex and/or yacc.

5.5.2 Usage

lex takes input syntax and converts it to C source code which can then be compiled and executed just as you would any other C program. lex creates a routine named yylex in the output file lex.yy.c. This can then be linked and executed to achieve the lexical analysis desired.

Figure 5.1 shows the process that is involved in creating a sophisticated program using lex and yacc. It is taken directly from *lex & yacc,* by Tony Mason and Doug Brown, O'Reilly & Associates, Inc. This is a great book and a must for any lex/yacc developer.

The figure accurately describes the process of generating both lex

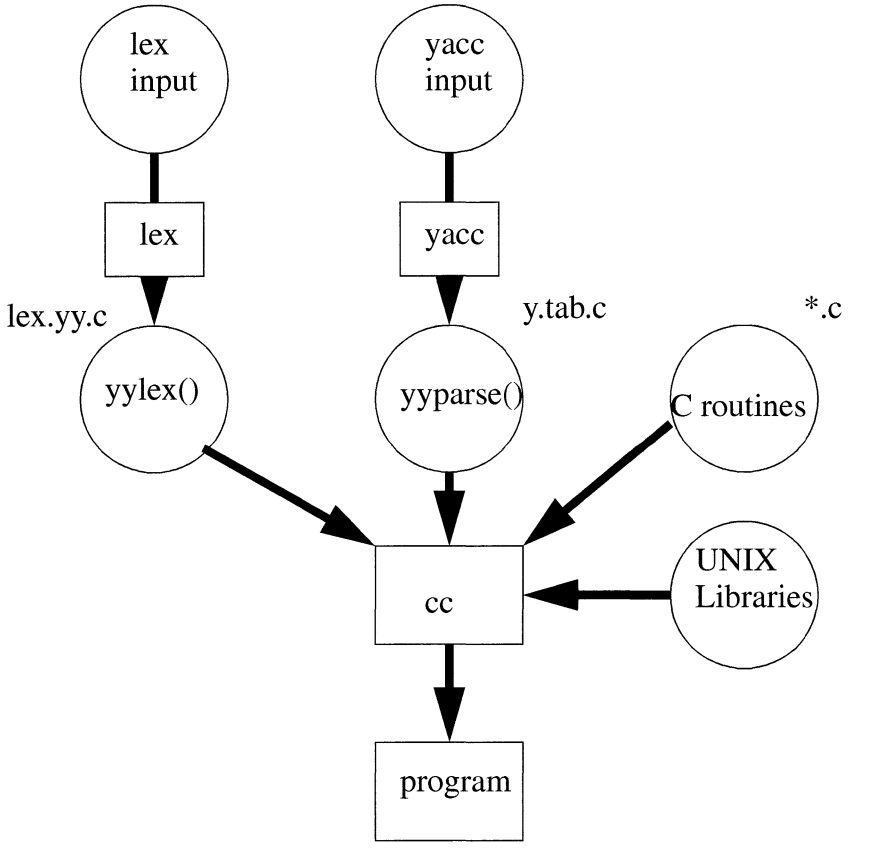

Figure 5.1 The lex and yacc process.

output (lex.yy.c), yacc output (y.tab.c) and your own C routines to combine into a single executable. This is a very powerful yet complex process which involves a large amount of knowledge about languages and parsers. Keep this in mind as you examine lex and yacc.

The basic synax for lex is:

```
lex [-fntv] [file ...]
```

where -f specifies faster compilation and larger scanner tables.
-n doesn't display output.
-t displays results on standard output instead of lex.yy.c.
-v is verbose mode.
file ... is one or more files to treat as input to lex.

The lex language. lex input files consist of sections based on rules and actions as well as on descriptions separated by %%s. The basic format is:

```
definitions
%%
rules
%%
user defined functions
```

The definitions section allows you to specify definitions which can define token values to pass to yacc as well as to other routines in lex and C. The rules section defines the rules and associated actions which your parser will use to parse and tokenize input. Finally, the user-defined functions allow you to write your own C routines to interface with the lex analyzer.

Often the user-defined functions consist solely of a main routine which takes some input and calls yylex. A simple example of this is:

```
extern int num;
...rules using num to determine behavior and possible
modifying num in the code...
%%
int num;

main(argc,argv)
int argc;
char *argv[];
{
if (argc != 0)
        num=atoi(argv[1]);
else
        printf("You need some command line arguments\n");
yylex();
}
```

This example demonstrates that you may create your own main routine and call yylex to do the lexical analysis. You are sharing the variable num, which you can access and modify in both the rules section and in the user-defined function section.

A slightly more complex example is given in the man page for lex as:

```
%% [A-Z]  putchar (yytext[0]+'a'-'A'); []+$;[]+putchar(' ');
```

This lex input file will replace all uppercase letters with lowercase letters and remove all blanks at the end of a line. Finally, it will replace all multiple blank occurrences with a single blank.

The [A-Z] is the pattern lex uses to match against. This means that a match will occur on any uppercase letter. The resulting C functions work as follows:

`putchar(yytext[0]+'a'-'A');`	Standard algorithm used to replace uppercase with lowercase letters by subtracting the difference between the ASCII representations of the characters themselves
`[]+$;`	Removes all blanks at the end of the line
`[]+putchar(' ');`	Replaces all multiple character occurrences with a single space

yytext[0] is a special lex function which provides the current tokenized string of information to output. If you don't use yytext, you will get the token value and not the token itself in the output stream. See the next example for this.

If you assumed the name of this file is uptolower.lex, you could execute the command:

```
$ lex uptolower.lex
```

The resulting file is named lex.yy.c and consists, on an AIX platform, of more than 572 lines of C code. The code is not included here for obvious reasons; however, take 1 minute and key this into your local system and execute the commands as listed above. You should get something similar.

Note that if you want to compile this system, you should use a command like:

```
$ cc lex.yy.c -ll
```

This will generate an executable a.out which can be executed normally and which uses standard input and output during its operation. Issue a command like:

```
$ ./a.out < datainput
```

and the a.out executable will modify the data input appropriately and will take the actions determined by the match to an uppercase character. Note that the -ll on the command line is necessary because the lexical analyzer yylex() needs a dummy main routine to initiate the program. Without the libl library, the program will not link.

Another simple example is the following simple lexical analyzer named caps.l:

```
%{
#define NUMBER 400
#define COMMENT 401
#define TE402
#define COMMAND 403
%}
%%
```

```
[ \t]+ ;
[0-9]+ |
[0-9]+\.[0-9]+        |
\.[0-9]+                        {return NUMBER;}
#*                              {return COMMENT;}
\"[^\"\n]*\"                    {return TEXT;}
[a-zA-Z][a-zA-Z0-9]+            {return COMMAND;}
\n                              {return '\n';}
#include <stdio.h>
main(argc,argv)
int argc;
char *argv[];
{
int val;
while (val = yylex()) printf("value is %d\n",val);
}
```

This example is taken directly from *lex & yacc,* by Tony Mason and Doug Brown, O'Reilly & Associates, Inc. It illustrates is the relative simplicity of the lex syntax. If you generated equivalent C code, it would need to be much more sophisticated and complex to provide the same function.

Anything between %{ and %} is copied directly into the resulting C source code. This allows you to embed any information in the declarations section that you want, and it will appear exactly as written in the resulting C source code file.

A thing to note about the lex syntax is that the first pieces of information on a line are simply UNIX regular expressions which allow you to express just about any type of expression without too much difficulty. This is the rules part of the lex input file and is designated by surrounding %%s. This specifies what you want to match and has an associated action to execute if a match occurs.

If you look at the above example and take a line to examine, you might use the following:

```
[0-9]+\.[0-9]]+    |
```

The [] denotes an exclusive choice of one character 0–9, and the + denotes one or more occurrences of the match. The \ is the escape character and ensures that the . (dot) is taken literally and not as a position holder to match any single character (as is the default case if the . is not escaped). The section [0-9]+ has the same meaning as the first occurrence. Therefore, this will match any string like:

```
1.1
0.0
11.11
100.100
...etc...
```

Note that this, taken in context with the other three lines representing different regular expression syntax, will match to a *number.* Note also that the | is a continuation line which allows you to specify alternate rules for each specific action. This is exactly what is done in this example to ensure that we get all the kinds of numbers we are looking for.

The other part of the lex input file is the actions section. Each rule has an associated action which will be performed when a rule is matched.

To use the above example, issue the series of commands:

```
$ lex caps.l
```

This will create a file lex.yy.c which contains all the C source code created from the lex input file. Next compile and link the code:

```
$ cc -o caps lex.yy.c -ll
```

This will create an executable caps which you can then execute as you normally would. If you took an input file named caps.input, which looks like:

```
This is a test of the caps (or CAPS) program. It has UPPERCASE
and lowercase words from which to CHOOSE. OKAY ...bye..
```

and ran the lex analyzer above, you would use a command like:

```
$ caps < caps.input
UPPERCASE
OKAY
```

Note that only the uppercase tokens followed by a blank, tab, or newline character willl be displayed.

The #define statements define tokens and associated token numbers to the parser routine yyparse. Both yacc and lex must share the same routines and tokens to ensure that the parser and lexical analyzer are communicating correctly and can recognize each other's tokens. See Sec. 5.6 for more details on parsers.

This section is a very brief introduction to lex. For more information and before you get serious about using lex, see the already referenced Nutshell book, *lex & yacc,* by Tony Mason and Doug Brown, O'Reilly & Associates; it is the de facto standard documentation for these tools and contains much more information than is presented here.

Flex and its differences. flex is a lex equivalent available from GNU software. See Sec. 7.6 for more examples and information on both lex and flex. In summary, the basic differences between flex and lex are:

flex generates faster code.

flex provides larger table sizes.

flex has no external library libl.a.

flex doesn't support RATFOR.

flex supports more flexible grouping with parentheses.

flex reads in one file; lex reads in many files and concatenates them together.

flex doesn't support lex input and output functions.

There are other functions that flex provides. See Sec. 7.6 for more information.

5.5.3 lex changes in AIX 4.1

There are some significant changes to lex in AIX 4.1 which need to be mentioned here. Much of the work done on lex is to bring it into conformance with XPG4; however, there are other changes which need to be mentioned.

The default type of yytext has been changed from an unsigned character array to a character array. This is only relevant for C++ users since char and unsigned char are the same in C. Beware of this.

lex in AIX 4.1 supports extended regular expressions as described in XPG4. Along with support for XPG4 regular expressions, multibyte characters are fully supported. In AIX 4.1,%z(%Z) is used to change the default number of multibyte character class output slots, while %x(%X) was used to accomplish this in AIX 3.2.*x*. Note that AIX 4.1 will support AIX 3.2.5 lex code with the appropriate changes related to %x(%X), and AIX 3.2.5 will run AIX 4.1 lex code except for %z(%Z).

As described on the IBM home page, "%x(%X) is also used in AIX 4.1 to specify an exclusive start condition. When the scanner is in %x state, patterns with no state specified will not be active as opposed to when in a %s state (regular start condition) where such patterns are active."

AIX 4.1 supports multibyte character sets with respect to character values. In AIX 4.1, ranges are evaluated based on the current collating sequence, including multibyte character sets. If the collating and the character sequences are the same for a given locale and range, this will have no effect on lex. However, with the inclusion of multibyte character sets and a renewed emphasis on locale, the output may vary between AIX 4.1 and AIX 3.2.*x* depending on sequence matching.

New delimiters have been added to lex in AIX 4.1. They are:

`[::]`	Character class
`[..]`	Collating symbol
`[==]`	Equivalence class

These new special delimiters allow lex to support character classes, collating symbols, and equivalence classes in addition to those supported in the earlier versions of lex.

Finally, lex under AIX 4.1 will no longer accept ranges where the end value is lower than the start value. If this occurs, an error will be produced and lex will exit.

5.5.4 Conclusion

lex is a very powerful language which supports many different programs. This section is realtively short because of a similar discussion which takes place in Sec. 7.6 as well as because of the sheer complexity and power of lex. flex is the GNU version of lex and provides enhanced functionality as well as greater speed and reliability. Because of this, it is recommended that you take a look at flex as an alternative to lex. More discussion of lex input files occurs in Sec. 7.6.

Some good examples of lex files exist in the groff distribution. See this distribution for working with lex input files and associated scanners.

5.6 yacc

5.6.1 Introduction

yacc is an acronym for "yet another compiler compiler." yacc is very similar to lex in that it provides a context-free grammar which it converts to C code which must be compiled and executed as you normally would C code. yacc generates a system called a parser. The parser is responsible for taking tokens generated from a lexical analyzer (typically) and generating an action based on the parsing of the input data. This is typically the second phase in the processing of an input stream and occurs immediately after the lexical analyzer has broken the input stream into independent pieces called tokens. The tokens are then analyzed for patterns and matches and processed according to actions defined based on the result of the parsing. If this seems complicated, it is. However, it is also extremely powerful and lends itself to techniques such as rapid prototyping and rapid application development. Using lex and yacc often speeds up the development of an application interface by orders of magnitude and is, in fact, used in many tools in AIX, including some on the accompanying CD.

yacc is commonly used to generate parsing routines in conjunction with lex. See Sec. 5.5 for more details on lex itself. There is a GNU equivalent called bison, which is discussed in Sec. 7.7. See this for more information on bison and other aspects of yacc, including differences between different implementations of yacc itself.

5.6.2 Usage

The basic syntax for yacc is:

```
yacc [-dvlt] grammar
```

where -d generates y.tab.h with the define statements that assign token codes with the user-declated token names.
-v is verbose mode. Creates file y.output, which contains table and conflict reports.
-l generates y.tab.c containing no numbered line directives.
-t sets y.tab.c so that it will be compiled with debug mode enabled.
grammar is grammar which determines the structure of the generated parser.

While this section does not discuss the yacc grammar in detail, there are some good examples in the distributions on the CD accompanying this book. See the groff distribution for some good examples of yacc grammar files and lex parser interaction.

You typically create a yacc input file named *.y. You also need a lexical program to pass tokenized information (yylex). Once you have created both of these, you run yacc on the *.y yacc input file to generate the C source code file. This file is typically named y.tab.c. Finally, you compile and link the resulting C source code files into an executable, which you can execute as you normally would.

The C source code file contains a routine yyparse(). This is very similar to lex's yylex and simply performs operations on tokens it gets from yylex(). Each time yyparse needs a token, it calls yylex and is passed the next token. yyparse() must be called from a main routine. Neither yacc nor lex generate a main function, and therefore you must link in with other libraries to complete the build.

The yacc language. While this section does not pretend to be a real reference for yacc, it outlines some very basic information and gives examples of yacc to illustrate the power of these tools. The basic structure of a yacc input file is very similar to a lex input file, namely:

```
declarations
%%
rules
%%
C routines
```

The declarations section consists of recognized keywords and associated definitions as well as C code. The basic keywords are:

%left	Declares left associative operators
%nonassoc	Defines operators which may not associate with themselves
%right	Declares right associative operators
%start	Defines the start symbol
%token	Declares token names
%type	Declares the type of nonterminals

Just as with lex, the rules section defines the grammar of the parser, and the C routines section contains any C source code you may want to include in your final C source code routine.

Again, a simple example file named print-int.y from the *lex & yacc* O'Reilly & Associates book is:

```
%token INTEGER

%%
lines: /* empty */
       | lines line
       { printf("= %d\n",$2);}
       ;
line:  INTEGER '\n'
       { $$ = $1;}
       ;
%%
#include "lex.yy.c"
```

The first part of this file contains a token definition INTEGER. This will be made available from the lex.yy.c lexical analyzer which is included at the bottom of the file. The %% denotes the beginning of the rules section of the file. The line lines: tells us that, as is the convention, the first of the two alternative definitions is empty. This is a yacc convention and says that an empty string is legitimate. The alternative definition (lines line) specifies that the input consists of one or more lines. Note that line is recursive and is in fact defined below as an INTEGER followed by a new line character.

The characters contained within the curly braces are the action items. These specify what should be done given a match of the rule. In the case of lines, a printf is specified which displays an equals sign and the integer on the input line. The line action sets the value of the return of the function to the first token. Note that yacc supports special syntax pertaining to the $ sign. $$ denotes the value that is returned from the action, while $n specifies the value of the *n*th token. Therefore $1 is the value of the first token, $2 the second, and so on.

Given this, the line action sets its return value as the value of the INTEGER on the input line and passes this back to the lines rule as the value line. The lines action takes the value of the second token (denoted by $2 and line in this case) and prints it to standard output. Fi-

nally, the included file lex.yy.c contains a lexical analyzer which must know about and determine what an INTEGER is.

If you assumed that you had both the appropriate yacc and lex files as described above, you would build and use them as follows:

```
$ yacc print-int.y
$ lex pint-int.l
$ cc -o print-int y.tab.c -ly -ll
```

There are several important things to notice about the above series of commands. First is that you can invoke yacc and lex in any order to create the resulting executable. Because print-int.y includes lex.yy.c, it is not necesssary to invoke lex.yy.c on the cc command line. Finally, it is necessary to include both the libl.a and liby.a archive libraries to ensure that all the proper yacc and lex files and a main routine are included so that the program will build properly.

This section does not do justice to the power of yacc and its possible interaction with lex. It does, however, present you with some of yacc's basic capabilities and should give you some idea as to how you can use a tool like this in your development work. There are several examples in the systems on the accompanying CD. See in particular the groff distribution for some very sophsiticated examples of lex and yacc files. Sun systems also have an example system in /usr/lib/yaccpar. Examine this for another good example of a yacc grammar file.

5.6.3 yacc changes in AIX 4.1

With AIX 4.1, multiple yaccs and multiple parsers in the same file are possible. The flag to specify an alternate parser to yacc is -y instead of -p. The -y option specifies a parser other than the default /usr/ccs/lib/yaccpar. The -p option is now used to change the prefix of all external names produced by yacc from the default yy to the prefix specified with the -p flag.

5.6.4 Conclusion

yacc, very briefly discussed here, is a very powerful parser generator which provides capabilities well beyond most AIX commands. The combination of lex and yacc provide a very sophsticated and powerful way to build complex parsing systems quickly and easily. This is how several available compiler systems were built. Keep these tools in mind for your development efforts.

Much of the functionality of yacc has been replaced and improved with a tool like bison from GNU. There is more information on some syntax and capabilities in Sec. 7.7.

Part

2

Nonnative AIX Developer Tools

While there are a wealth of tools available on the native AIX platform which provide tremendous power and flexibility when using AIX, there are just as many if not more power tools that don't come native with an AIX platform. Some of the more powerful tools are documented in this section.

Editors, compilers, debuggers, object-oriented compilers and analyzers, text formatters, and many native AIX replacement tools are just some of the tools discussed in this section of the book. Keep in mind that a large portion of this part of the book is dedicated to GNU tools. GNU is reengineering most of the tools that come with native AIX and, in fact, is rewriting a freely distributable UNIX kernel based on the Mach microkernel architecture from Carnegie Mellon. Many of the tools discussed are part of the GNU paradigm which is attempting to replace UNIX with a freely distributable version of the same environment.

Tools such as GNU's C compiler, called gcc, are considered better than what you can typically get from your vendor. Unlike many shareware and free tools you can get for DOS and Mac platforms, these tools are of the highest quality. Support is usually excellent, and with a price of $0 for most products discussed in this book, the price is right. Keep in mind that all of these tools are freely available from the Internet as well as in the distribution that accompanies this book.

Chapter

6

The Internet

UNIX and the Internet used to be synonymous; however, with the explosion of growth on the Internet, many other types of machines are beginning to access the Internet. Because of the correlation between UNIX tools distribution and the Internet, this chapter is necessary to present a background on the history and distribution of software tools for UNIX.

The Internet is now being used to deliver not only AIX developer tools but MS-DOS, Macintosh, VMS, and many other types of power tools. The Internet is already the information superhighway that many people are talking about today. This chapter presents just enough of the capabilities and power of the Internet to prepare you to get information from it and to know where to learn more about it. This book by no means attempts to be a reference guide to the Internet but instead attempts to provide just enough information to allow you to get AIX tools and software packages as they become available.

6.1 What Is the Internet?

The Internet is simply the largest computer network in the world. With millions of nodes, it surpasses even the largest of corporate networks by many orders of magnitude. It is a collection of many networks bridged and gatewayed together to provide a relatively seamless network which spans the entire world. At one time the Internet was a large network consisting of the IP protocol exclusively; however, today it consists of many networks and protocols gatewayed together to form a patchwork of networks that span the world.

The Internet started out as the ARPAnet, which was an experimental network designed to support government research for the Defense Department. Because of the disparities of hardware and software plat-

forms among the participants, there was an immediate need to develop an architecture which supported all systems. The protocol known as Internet Protocol (IP) was invented as an ISO layer 3 protocol that would run on both LANs and WANs while the ISO organizations debated their own layer 3 protocol.

Berkeley began distributing IP with their UNIX kernels and an entirely new network protocol de facto standard was born. Along with Berkeley, a number of new agencies were beginning to build computing centers, and the only economical way for them to connect everyone together was through the ARPAnet since most universities and research labs were now on it.

In the mid 1980s, the National Science Foundation (NSF) put a 56-kbs architecture in place based on regional supercomputer centers that ran IP. This not only allowed universities and research labs access to their network (known as NSFnet) but to access the ARPAnet as well. The NSF is promoting educational access and helping defer costs in an effort to place all educational institutions on the NSFnet. This has been extremely successful, and as a result, demand is growing dramatically not only in the educational market but in the commercial one as well. As students leave their educational environments and move to the commercial world, they are convincing corporatons to connect to the Internet.

There are several volunteer groups which govern and maintain the Internet. The presiding body responsible for guiding Internet development and direction is the Internet Society (ISOC). They appoint members to a group known as the Internet Architecture Board (IAB). The IAB is responsible for setting standards for communications and architecture to sustain the tremendous growth seen in the Internet in the last few years. The IAB meets regularly to discuss these issues and hear proposals for change and growth.

Finally, the Internet Engineering Task Force (IETF) is responsible for short-term goals and deliverables on the Internet. They hold regular meetings, and everyone who is interested in the Internet is invited. By splitting into working groups, different problems are solved based on volunteers' experiences and interests. The typical output of a working group is a report to the IETF and often the IAB to be set as standards.

Because of the regional nature of the Internet, a Network Operations Center (NOC) was established to handle each region or backbone. For example, NASA has a set of backbones which are managed by one NOC, while the NSF backbones are managed by another NOC. If you connect to the Internet, you will get a NOC that is responsible for your connection. If it cannot solve your problems, it escalates to the next NOC that can solve them.

Beyond providing the NOC, most agencies are responsible for paying for their segment of the Internet (e.g., NASA, NSF). The Internet was developed to share research and engineering data and not commercial data; however, this is changing as corporations and individuals realize the potential of the Internet for sharing information quickly and inexpensively. As this commercial access has grown, several companies have sprung up to provide for-fee commercial access to the Internet. These companies purchase a segment of the capacity of the Internet and resell access to commercial customers. Companies like PSInet, Netcom, and UUnet provide commercial users access to the Internet. The price ranges from just a few dollars a month for a simple dial-up line to thousands of dollars a month for full-function leased-line capacity running IP.

The Internet is simply an extension of the phone and leased line infrastructure around the world. There are gateways and routers which connect a large number of disperate networks around the world. As Fig. 6.1 denotes, you can access the Internet with a variety of components including modems, routers, and bridges. The Internet uses both the public telephone network and a variety of leased lines from a vari-

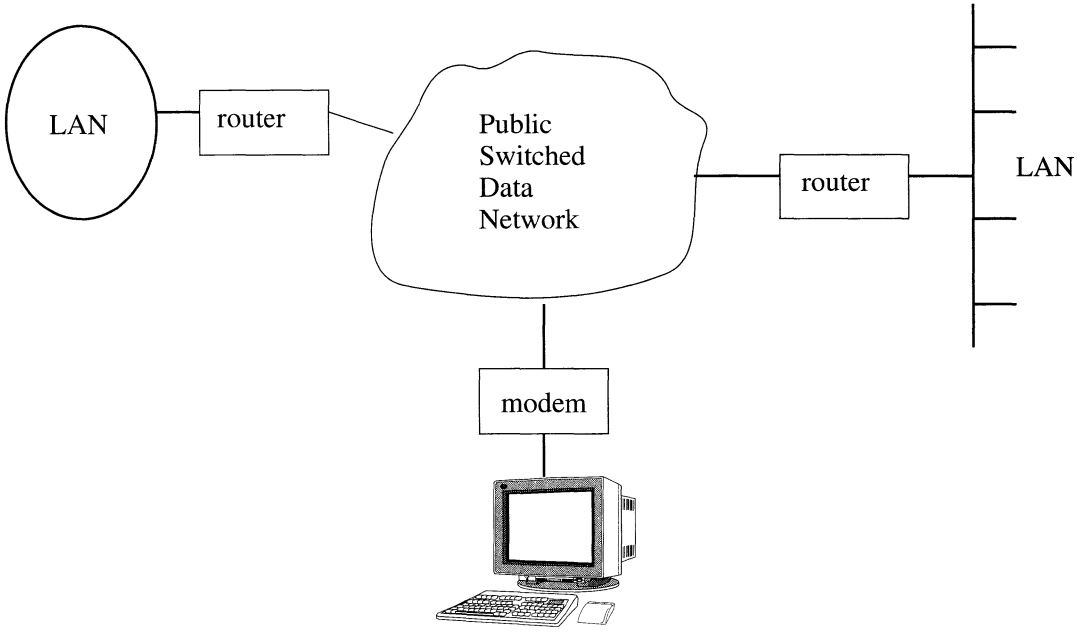

Figure 6.1 The Internet.

ety of government organizations and private access providers such as UUnet.

There are protocols which run between the routers and gateways which allow each to understand who they are responsible for based on a concept called a domain. A domain is established by a place called the Network Information Center (NIC). The NIC has been responsible for distribution of addresses and domains since the inception of the Internet.

Internet addresses are IP adresses that are represented by four groups of numbers separated by periods. An example of an IP address is:

```
191.9.200.1
```

Each number left to right represents a subnetwork of the higher-level representated to the left. In this example there is a large network represented by 191. Within this network there is a subnetwork represented by 191.9. Inside this is 191.9.200, and finally the actual node IP address is 191.9.200.1. The NIC is responsible for providing IP (e.g., Internet) addresses based on dividing customers by geography or organization. These numbers allow routers and gateways to section off certain portions of the Internet to reduce the packet loads on individual segments of the network. For example, if you want to send a message to address 192.10.10.10, the initial router you contact through your part of the Internet maintains a table containing who is responsible for the 192 subnetwork and then figures out the quickest way to get your infomation to this subnetwork. It may have to route it through several gateways and routers; however, the algorithm used by the routers will ensure the most efficient path at the time it calculates this. It is important to note that this path may be different at any given time based on network congestion and router/gateway availability.

Figure 6.2 denotes a network which consists of a gateway machine between two networks. If a packet from the 191 network wants to get to the 192 network, it must be routed through the gateway. Note that the gateway can be a UNIX computer or a specialized device such as a router or brouter (bridge/router). This device contains tables which help to route the packets to the appropriate network based on the IP address. The Internet works very similarly to this. A rough schematic is shown in Fig. 6.3. This figure represents the telecommunication links between the interior gateway routers. Most of these lines are high speed T1 and T3 lines, but there are still lines which run at much slower speeds on the Internet network. Suffice it to say that you must connect to the Internet network either directly through a router that is directly connected to the Internet backbone or through an access

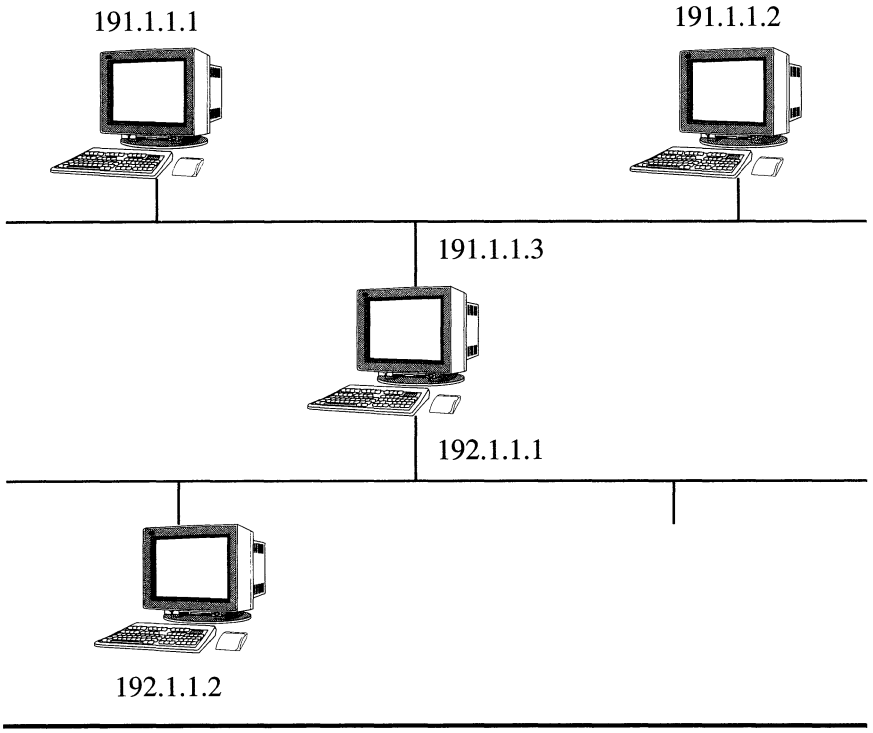

Figure 6.2 IP gateway.

provider such as UUnet or PSInet. The Internet itself takes care of routing packets through the network by communicating between all routers on the network. This ensures redundancy and reliability on the network and increases the stability of the entire network.

Because remembering IP addresses is virtually impossible for people, the NIC began to give computers names people could remember. Originally, the NIC distributed all IP addresses and names; however, as the Internet began to grow, this became impossible. A new scheme known as the Domain Name System (DNS) was developed. This gave each group of users on the Internet the responsibility to manage its own machine names and IP addresses. The naming structure was based on the success of the IP addressing scheme and therefore was divided into subgroups separated by periods. Each level in this system is called a domain. An example is:

```
fnalb.fnal.gov
nic.ddn.mil
fred.csu.upenn.edu
```

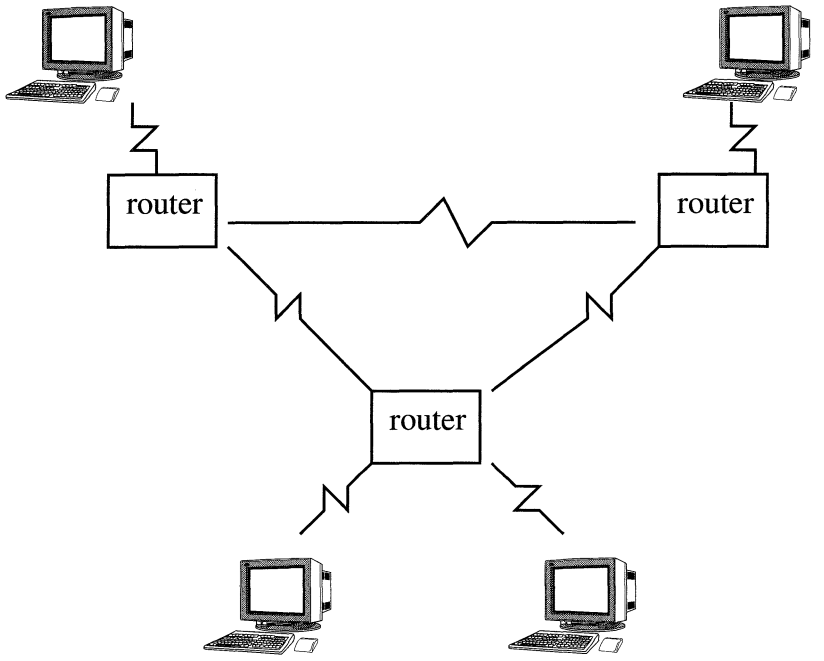

Figure 6.3 The Internet and its routers.

This works in exactly the opposite way from the way IP addresses work. The major domain is the farthest to the right in this address (known as domain name). There are six main domains: gov (government), mil (military), com (commercial), net (network resources), org (other organizations), and edu (educational institutions). There are new domain names emerging based on the country of the machine (e.g., au, us, bg, etc.). These will become more prevalent in the next few years. Within a given domain, there exist machines and groups of machines. For example, take the address:

```
fred.cs.uchi.edu
```

The machine exists in an educational institution, probably at the University of Chicago (uchi), perhaps in the computer science department (cs), and the actual machine name is fred, who is probably the graduate student who uses the machine. This domain name may translate to an address of 192.9.200.1. There is a correlation between the uchi and the subnetwork addresses of 192.9 since the NIC probably gave the University of Chicago all addresses within 192.9. This correlation is main-

tained by the NIC and is crucial since neither IP addresses or domain names can be duplicated on the Internet. If this occurs, major problems result, and therefore it must be avoided at all costs.

You can have two machines with the same name, but they must be in different domains. For example, fred.cs.upenn.edu would be perfectly fine. Even fred.bio.uchi.edu is find as long as they are not exactly the same. Note also that the IP addresses of these machines will be very different depending on their domains. Most likely fred.cs.upenn.edu has a different major IP address, while fred.bio.uchi.edu probably starts with 192.9.

6.2 Tools of the Internet

Note that this is not entitled "Tools on the Internet" but "Tools of the Internet." These are tools that allow the Internet user to be most productive and reap the most benefit from the Internet. There are really three fundamental tools of the Internet which enable users to maximize its usefulness:

1. News
2. anonymous ftp
3. mail

6.2.1 News

News is the Internet bulletin board system. News consists of groups of information and exchange based on a topic known as newsgroups. Newsgroups consist of readers that present a menu-like interface to newsgroups and allow you to browse and interact with any newsgroups of your choice.You get information on "newsfeeds" which consist of many megabytes of information per day depending on which newsgroups you intend to read. The full newsfeed daily traffic is in the hundreds of megabytes per day and is a tremendous effort to manage. Most users subscribe to a small subset of newsgroups in an effort to minimize the traffic as well as management aspects of the newsfeeds.

Newsgroups are hierarchical in nature, consisting of what look very similar to domain names. For example:

```
comp.os.aix
```

The largest group is the comp (computer) group followed by the os (operating system) group and finally the aix operating system division of this newsgroup.

By far the largest set of newsgroups and information comes from the Usenet. The Usenet consists of seven primary newsgroups:

1. comp—computer science and related topics
2. misc—miscellaneous
3. news—questions relating the news network itself
4. rec—recreational activities
5. sci—scientific and research activities
6. soc—social activities
7. task—controversial topics such as religion, politics, and the like

There are other sources for newsgroup feeds; however, they are beyond the scope of this book. See App. D for more books and information on this topic. In addition to the core Usenet newsgroups, there are several alternative news groups including:

1. alt
2. bit
3. biz
4. ieee
5. gnu
6. k12
7. vmsnet

These and other newsgroups like them contain much lively discussion and information on an almost infinite variety of topics. You should carefully choose which newsgroups you would like to receive since you will quickly be overwhelmed by information if you aren't careful. Finally, other sources, such as Clarinet which is run by the United Press International (UPI) and contains up-to-the-minute news and information and broadcasts, are available on a for-fee basis.

To get a better understanding of the Usenet network, you must examine its origins. When the Usenet began, it consisted of people using primarily PCs connected with modems. They would periodically dial each other up and trade information. As the group grew, it developed methodologies for sharing information which involved known locations and flows for the shared information. A person or persons agreed to be the distributor of certain types of information by allowing others to dial into their computers and download information that they required. This was known as a newsfeed. As this matured and the number of newsfeeds grew, a large network consisting of a complex and confusing

architecture evolved. With the adoption of high-speed WAN technology, much of this transfer is now occurring over leased lines. However, there are still many informal newsfeeds occurring around the world.

Figure 6.4 is from *The Whole Internet User's Guide & Catalog,* by Ed Krol. It is representative of the kind of network that News typically travels through. News servers originally used the basic telephone network to transfer information. Today, many newsfeeds move across the Internet at T1 and T3 speeds for faster transfer.

Different news servers provide different newsfeeds based on the needs and usage of that particular node's users. Administrators must choose which newsfeeds they wish to accept and those they wish to distribute. If administrators choose to distribute a newsfeed, they accept the responsibility that others may call and wish to connect to them, and they need to be responsive and provide those requesters access.

By providing transparent network access to users on the network, the newsreader can access newsgroups transparently. The newsreader has the responsibility not only to act as the menu-like front end but to attach to the news server and download article lists as well as actual articles selected. It acts as the router to control access to newsgroup information. There are several prominent readers:

1. nn
2. rn
3. trn
4. tin

Since the man pages for each of these tools are some 40 to 50 pages, this book will not discuss individual readers. Suffice it to say that

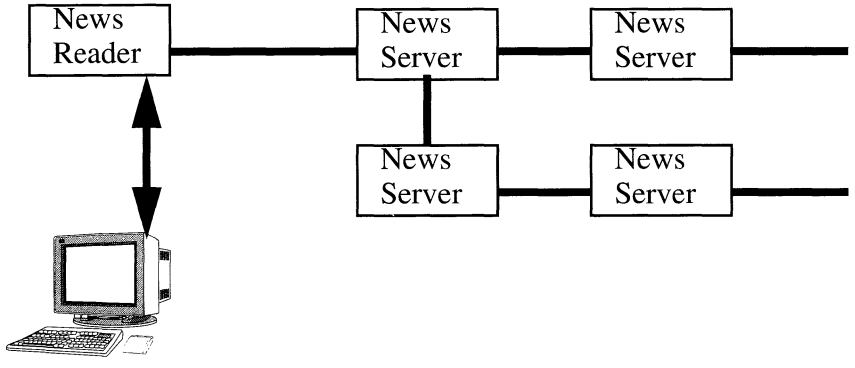

Figure 6.4 The News network.

reader choice is like religion, and each person prefers one or the other. Each reader has disadvantages and advantages, but nn is certainly as good as any other and provides certain benefits such as more control when reading information and tracking. You can change readers at any time, so feel free to experiment with each of the readers.

6.2.2 FTP

The second core technology of the Internet is the anonymous FTP capabilities. FTP stands for File Transfer Protocol and is the core utility that works with TCP/IP to provide file transfer capabilites between nodes on a network. By utilizing login security checks, users can transfer files based on the existence of their accounts on each machine.

The general syntax of the FTP command is:

```
$ ftp remote-machine-name
```

where remote-machine-name is the name of any other machine that you can access independent of where the machine is or what operating system it runs. As long as it supports the FTP protocol, you can use FTP to transfer files.

When a connection is established, you are prompted for a username and password for the remote machine. After typing these in, you are placed at the interactive FTP prompt ftp>. Once you are at this prompt, you are ready to begin transferring files. An example of a file transfer from your current machine named dumpy to another machine named dopey is:

```
dumpy>$ ftp dopey
Connected to dopey.
220 dopey FTP server (AIX(r) System V Release 4.0) ready.
username: kevin
Password required for dopey.
passwd: password
User kevin logged in.
ftp> put /tmp/stuff
200 PORT command successful.
150 ASCII data connection for stuff (55 bytes).
226 ASCII transfer complete.
83 bytes sent in .02 seconds (4 kbs)
ftp> quit
221 Goodbye.
dumpy> $
```

This example transfers a file named /tmp/stuff from a machine named dumpy to a file in your home directory named stuff on a remote machine named dopey. FTP allows for file transfer in both directions. To transfer a file from a remote machine named dopey to the local machine named dumpy, use the get command. For example:

```
dumpy>$ ftp dopey
Connected to dopey.
220 dopey FTP server (AIX(r) System V Release 4.0) ready.
username: kevin
Password required for dopey.
passwd: password
User kevin logged in.
ftp> get /tmp/stuff
200 PORT command successful.
150 ASCII data connection for stuff (55 bytes).
226 ASCII transfer complete.
83 bytes sent in .02 seconds (4 kbs)
ftp> quit
221 Goodbye.
dumpy> $
```

There are a variety of commands within FTP that may be of some use; they are:

`binary`	Transfers a file in binary mode and performs no translation
`ascii`	Transfers a file in ASCII mode; performs any necessary translation to maintain an accurate representation
`put local-file-name remote-file-name`	Puts file from your local machine to the remote machine
`mput filelist`	Multiple file put
`get remote-file-name local-file-name`	Gets a file from the remote machine to your local machine
`mget filelist`	Multiple file get
`user username`	Logs on to remote machine
`help`	Prints list of all FTP commands
`lcd dir`	Local change directory
`cd remote-dir`	Changes remote directory
`close`	Closes current ftp session on remote machine and returns to FTP mode
`delete remote-file-name`	Deletes remote-file-name on the remote machine
`dir filename`	Shows remote filename listing (wildcards supported)
`open machine`	Opens a connection to a remote machine
`pwd`	Prints the name of the current working directory
`quit`	Closes any open connections and exits FTP

If you are interested in transferring multiple files with the mput and mget commands, you will be prompted for each file before transfer unless you invoke FTP with the -i switch. Keep this in mind before you invoke FTP for file transfer.

Also, to interrupt file transfer you can use the CTRL-C sequence. This will halt file transfer after the server has had time to process the interrupt. It usually takes some time before the data stops flowing into or out of the local machine based on network load and file server performance and load. Be patient.

You will note the lack of any cat command in the FTP subcommand listing above. If you want to examine a file before transfer, you must be a little clever and remember AIX and its standard I/O structure. To examine a file on the screen before transferring it, use:

```
ftp> get file1 -
```

where the - represents standard output. If this doesn't work, try:

```
ftp> get file1 /dev/tty
```

which uses the standard terminal device name as the output file. Remember, everything in AIX is a file, and, as such, redirection is as simple as that shown above.

6.2.3 Anonymous FTP

The FTP utility relies on username and passwords for ensuring integrity of filesystems and machines. However, there is a de facto standard in AIX and on the Internet for using FTP for general-purpose file sharing and transfer. It is known as anonymous FTP. This means that when you connect to the remote machine and are prompted for your username, you type anonymous. The password can be left blank; however, it is generally accepted that you type your mail address so that you can be tracked and notified in the case of a problem. After logging on, you typically have access only to a small area of files which are explicitly permitted to you. This is how almost all Internet software is shared and propogated. All FTP commands are maintained as in formal FTP; the only difference is that now you don't have to have an account on a machine to access its files. This allows millions of users to share ideas and software without requiring any maintenance and overhead associated with managing large groups of users and software systems. Anonymous FTP is the way a general user will interact with the Internet for almost all tasks.

6.2.4 Internet mail

Electronic mail (e-mail) to the Internet is one of the primary benefits of the Internet. Beyond the capabilites of file transfer via anonymous FTP and telnet capabilities, which allow for machine sharing, e-mail is by far the most useful feature of Internet access. By acquiring Internet access, you gain the ability to send mail to millions of computer users, not only those directly attached to the Internet but through gateways to other networks such as MCI Mail and CompuServe as well. Mail is clearly a great way to communicate and exchange information without

requiring expensive and high-maintenance options such as leased lines and IP access to the Internet.

Limitations of the e-mail system are time and cost. It clearly costs you something to send a message to someone on the Internet; however, the cost is a matter of cents and depends on where you send the information and how large the message is. Suffice it to say that the cost is very low compared to the increased functionality received. It also costs you something in terms of time. The Internet is a store and forward network, which means that packets of information are gathered at a node and then, at some later time, forwarded to the next node in the chain. There may be zero to many nodes in between your sending node and the receiving node depending on the physical topology of the networks and node configurations. This is transparent to the end user since the underlying mail subsystem handles the routing of this information; however, it does affect the transfer times of the mail.

The store and forward nature of the Internet mail subsystem means that information such as mail messages may take some time to reach their final destination. While the time required to receive a message from the Internet is usually seconds, it may take minutes or hours for a message to reach its final destination depending on the status of links and machines on the path from initator to final destination. This is clearly not as convenient as using an interactive access method in some instances; however, the many conveniences of mail often outweigh the costs.

Addressing. Mail addressing is based on the domain naming conventions described earlier in this chapter. The general format of a mail address is:

```
usename@machine-name
```

where username is the usename of the person to whom you want to send mail, and the machine-name is the hostname of the machine. For example, to send mail to a user named kevin on a machine named devtech.devtech.com, you would structure the mail name as:

```
kevin@devtech.devtech.com
```

This uniquely identifies me on the Internet, and it is guaranteed that there is no other person with exactly that same address on the Internet. You do not need to worry about including any other information than the username and node name to which you want the mail sent. There may well be nodes that this information will pass through be-

tween the authoring node and the receiving node, but this is hidden from you by the Internet mailing subsystem.

To send a mail message to the above user, you could use the following:

```
$ Mail kevin\@devtech.devtech.com
Subject: test
this is a test of the Internet mailing stuff. Please reply if you
get this.
thanx...kevin
.
```

There are several things to note about the above mail. First note that the @ is preceded with a backslash. This is because the @ is a special shell character and should be "escaped" to avoid shell translation. Remember, in AIX, the shell interprets the entire command line before passing it to the program to be executed. In this case, if you do not precede the @ with a \, the shell will interpret the @ as a shell metacharacter and will perform actions before passing the command line information to the Mail executable. To avoid this, the @ must be "escaped" and thus avoid interpretation. Another thing to note is that all mail is terminated with a . in the first column on the last line of the text input area. This will take you out of text input mode and back into command mode. Finally, note that the command Mail was used to invoke a mail subsystem. Remember that AIX is case sensitive, and, in this case, there is a difference between mail and Mail. The Mail interface is a Berkely interface which is slightly more feature rich than the mail interface; however, it does not matter which interface you use to address Internet mail.

There are many gateways to other networks beyond the standard Internet such as MCI Mail and CompuServe which you may want to access. Your access provider will supply information on how to address mail to ensure that it reaches those connected networks. There are a few standard naming and addressing conventions, however, that allow you to send mail through gateways to these networks:

Bitnet. Bitnet address are of the form user@host.bitnet. To route from the Internet to a node on a Bitnet network, use user%host and follow this with a standard gateway from Internet to Bitnet. For example:

```
kevin%fnal@fnal.fnal.gov for the Bitnet address kevin@fnal.bitnet
```

Ask your access provider for more information on how to access the Bitnet network.

CompuServe. Standard CompuServe addresses consist of two numbers separated by commas (12345,123). To route to CompuServe from the Internet, use the syntax:

```
12345.123@compuserve.com
```

Note that the comma used in CompuServe addresses is replaced with a period and followed by a hostname of compuserve.com.

MCI Mail. MCI Mail uses two different kinds of addresses: number and name. If you are sending to a number, simply following the number with the standard hostname syntax for:

```
1234567@mcimail.com
```

If you are addressing a name, use the syntax firstname_lastname followed by the hostname:

```
Kevin_Leininger@mcimail.com
```

There are other networks which you may want to access. Ask your Internet access provider for more information on how to access them.

Sending binary data. Most mailers are structured to only send text files. If you are interested in sending nontextual information such as executables, audio files, etc., you must encode these files before correct transmission can occur.

The utilities used most often to accomplish this are uuencode and uudecode. uuencode is provided on most AIX platforms and converts binary information to text. Once you have converted the binary information to text, you can then mail it as you normally would any text file. Then the receiver must uudecode the file before accessing it. For example, to send an executable file named runit to user joe on hostname galaxy@devtech.devtech.com, use:

```
$ uuencode runit < a.out > runit /*takes a.out and makes a text file
runit*/
$ Mail joe@devtech.devtech.com
Subject: uuencoded program
Here is the runit program, have fun...
~r runit /* mail command to include a file*/
.
```

This creates a file with the first line text and the uuencoded file following. When joe receives the mail, he will see something like:

```
$ Mail
begin 644 runit
```

Joe first must save the mail message as an external file and then run uudecode on it. The uudecode command will ignore any information before the begin command and will rebuild the executable to be exactly as before it was uuencoded. Joe now has a program named runit which he can execute normally.

Note that for an executable, he must be running on the same type of machine as the executable file was originally built on. uuencode and uudecode do not provide binary compatibility but are merely dumb tools to allow for binary transmission from one machine to another.

It should be noted that there are a variety of standards discussions occurring with respect to sending binary files, and there will be standards for mailers and binary files adopted in the very near future related to the MIME extension. MIME stands for Multipurpose Internet Mail Extensions. This represents the new format in transmitting multiformat files between dissimilar computers.

Returned mail. There are many reasons that you may receive returned mail. The address was incorrect, the receiving node was down, the remote host was unknown, or the remote machine is not configured correctly. Any one of these will cause mail to be returned with a message in your mail file like the following:

```
$ Mail
Subject: returned mail for bogusid
Status: R

Mail error was: 550 <bogusid>... User unknown

-- returned mail follows ---
To: bogusid
Subject: test

this is a test
```

If the remote host was unknown, you will receive all the information in the returned message which will document the path the mail message took before being returned. An example is:

```
Subject: returned mail for bogusid
Status: R

Mail error was: 5110 <host!bogusid>... Unknown host

-- returned mail follows ---
To: host!bogusid
Subject: test

this is a test
```

By tracking the path of the returned mail, you can see which machine failed in the store and forward mail network. Check the hostname and

full address; if it is correct and the mail will not go through, contact you Internet access provider with a detailed description of the problem.

You will also see a returned mail message with the "User unknown" message. Check that this user does in fact exist; if he or she does, contact your Internet access provider for more information.

If there is a significant downtime of the receiving machine, you may see returned mail with a Subject line describing the condition. If this occurs, contact that machine's system administrator for more information. Note that due to the store and forward nature of the Internet mail subsystem, your mail message will be retransmitted several times to the receiving node before failing. This will avoid having this kind of problem with a short-term downtime for a machine.

If you send mail to more than one person and you get a returned mail message, check to see who received the mail and who didn't and resend to only those who did not receive the message. All mail messages that can be sent will be, and only those with unreachable addresses will be returned.

Getting files with mail. If you don't have direct access to the Internet and want to get files and archives, you can use mail to accomplish this. While interactive methods such as FTP and telnet are more convenient, there are a variety of reasons for using mail to access and retrieve files.

There are two general kinds of mail servers available via e-mail:

1. Generic Internet file servers
2. FTP mail servers

Generic Internet mail servers provide a mail server daemon which waits for incoming mail with a specific subject or text heading addressed to it and process the information contained in the mail message. Depending on the included information, it performs different actions from sending help information on its commands to getting files from a local directory and sending them to the mail originator. The standard syntax for these servers is to include a command on the Subject: line such as:

```
$ Mail mail-server@devtec.devtech.com
Subject: help
.
```

which will provide help on the commands and capabilities of the mail server. Note that text for the message is not required since only the Subject: line is examined to determine an action for the mail server. The other common command is send. The syntax is:

```
Subject: send filename
```

where filename is a filename which may include a relative or fully qualified directory name. For example:

```
$ Mail mail-server@uunet.uu.net
Subject: send ls-lR.Z
.
```

will retrieve a file named ls-lR.Z from the machine named uunet.uu.net. This file just happens to be the compressed index of all files on the UUnet machine.

The FTP mail server provides the capability for a remote machine to FTP a file to itself and then mail it to your machine. This provides unlimited access to Internet archives through one of these FTP mail servers. The FTP mail machine most commonly used is decwrl.dec.com. To address commands to this machine, use:

```
$ Mail ftpmail@decwrl.dec.com
Subject: junk
connect devtech.devtech.com
chdir pub/physics/hep
binary
get intro
quit
.
```

As shown above, you build an FTP script in your mail text which is executed on the decwrl machine. In the above script, decwrl connects to devtech.devtech.com, changes directories to the pub/physics/hep directory, sets mode to binary, and transfers a file named intro back to your machine via e-mail. The Subject: line is ignored with this server, but you can use it to document your activities. There are many commands available on this server, and you should send the command help to the server for complete details.

6.3 Who Uses the Internet?

As the Internet grows and gains more commercial users, questions arise about who is paying for traffic on different parts of the Internet. What began as an educational and research network is rapidly becoming a commercial network. When you establish a connection to the Internet, you must specify whether you will be using it for research and education or commercial traffic. If you choose research and education, you will be routed over prefered routes using government subsidized links on what is known today as National Research and Education Network (NREN). NREN was recently federally funded to allow all In-

ternet research and education users to access a common backbone. However, if you establish your access as commercial, you will need to use a value-added Internet access provider such as Performance Systems International (PSI) or UUnet. See App. E for more information on how to contact these organizations.

Most computer hardware and software vendors are beginning to use the Internet for distribution of software updates and patches. Because of the lower cost of distribution and of access to more timely information, the Internet makes the most sense for them. Of course, the biggest users are still research laboratories and universities; however, commercial users are beginning to utilize the Internet for collaboration between their engineering and research businesses as well as other parts of their organizations such as marketing and sales. The complexion of the Internet is changing rapidly, driven by the commercialization of AIX and the entire open systems paradigm. The most fundamental change in the Internet is the Worldwide Web (WWW). This, along with a browser known as mosaic, is revolutionizing the Internet. See Sec. 6.8.5 for more information on WWW.

6.4 Why Use the Internet?

The question is really not, "Why use the Internet?" but, more importantly, "Why not use the Internet?" The Internet provides tremendous access to information that is vital to the efficient operation of computers and computer specialists around the world. Because of this, the Internet is rapidly becoming the most used of all computer networks.

Actually, there are three primary reasons to use the Internet: (1) access to information newsgroups, (2) access to software via anonymous FTP, and (3) access to the WWW. These have been discussed, and based on discussions in the rest of this chapter, we will see why the Internet is such a powerful resource and why you will use it more and more in your daily life.

6.5 How to Access the Internet

There are a variety of services that provide both partial and full access to the Internet. As was discussed before, you must decide what kind of access you need before contracting with a service provider for Intermet access.

There are rules that must be agreed to and forms that must be filled out before access to the Internet will be allowed. If you choose dedicated Internet access, you will be assigned an Internet addresses and a domain based on information you give to your service provider. This information will be registered with the NIC and will be made available

on the Internet worldwide through the primary routers and gateways. This allows anyone on the Internet to route to you with things like anonymous FTP and mail. Keep this in mind when you talk to your service provider, and make sure you have thought through the ramifications of the "world" having this kind of access to your machine. This is not intended to scare you away from getting access to the Internet. There are many ways to secure your network against network violations; however, depending on how you access the Internet, you may be opening up an entirely new set of issues about the security of your machines and networks.

6.5.1 Dedicated Internet access

For dedicated Internet access, you will pay several hundred or perhaps thousand dollars for a company to provide you a leased line and a router/modem device at your site. This typically consists of a Telebit Netblazer device, which is the standard in the AIX arena for high-speed modem communications. You will also incur fairly high costs relating to leasing the line, probably $1000 or more per month depending on line speed.

Dedicated access provides you with full IP capabilities and is clearly the highest-function solution available. This means anonymous FTP and complete peer-to-peer access to other machines on the network. You will have the Internet as a WAN which looks just like your LAN.

6.5.2 Partial Internet access

There are several kinds of partial Internet access ranging from Serial Line Interface Protocol/Point to Point Protocol (SLIP/PPP) solutions, which look very similar to dedicated access, to UNIX-to-UNIX Copy Program (UUCP) access, which is much lower function.

SLIP/PPP. SLIP allows for an IP data stream to run over standard phone lines. You can use SLIP to run a LAN protocol over a standard phone line and use tools like telnet, FTP, NFS, etc., just as if you were connected to a higher-speed LAN such as token ring or Ethernet. SLIP is freely available on the Internet and is ported to virtually every platform. You may contact a service provider listed in App. E for more information on SLIP access. PPP is the successor to SLIP and contains SLIP functionality and much more. PPP is a better solution if your access provider supports it.

The SLIP/PPP solution will allow you to look as if you have a dedicated connection to the Intenet only when you bring SLIP/PPP active. This allows you to avoid the cost of a leased line while still realizing all the benefits of running as a dedicated peer-level node on the Internet.

Typical costs of this are a few hundred dollars a month for unlimited access from a service provider. There are other packages which have lower fixed costs and associated connect time charges. Talk to your service provider for more details.

UUCP. UUCP is a tradition in UNIX; it was invented to allow UNIX machines to transfer files over phone lines. It later added support for remote login and became the WAN protocol of choice between UNIX machines over phone lines. As high-speed networks proliferated, UUCP became much less commonly used; however, it is still used by many Internet access providers for a class of Internet connection known as UUCP access.

The cost of this kind of connection is tens of dollars a month and an inexpensive modem. However, you do not get any IP capabilities and therefore don't get anonymous FTP and WWW capabilities. You can get News as before, but it must be set up with UUCP for polling on a predefined schedule. All access to files is done through mail. You must send mail to a mailserver on your access provider's machine, and they are responsible for sending your requested files to you in the form of mail. You must then use mail to manipulate and control your files from the Internet. This is clearly a lower-function way of interacting with the Internet; however, it does work and is certainly the lowest-cost solution available.

6.5.3 Gateways

There are other ways to gain access to the Internet such as Bitnet, CompuServe and MCI Mail. If you have accounts on these networks, you have mail access to the Internet and can send mail and exchange information via gateways between these networks. However, you do not have the kind of access that is provided in the other access methods, and this should not be your primary interface to the Internet if you intend on using it in any serious way.

6.5.4 Listing of service providers

There is a listing of service providers for Internet access in App. E of this book. As you can see, there are many of them, and they provide many different types of services. Contact them directly for more information about their products and services.

6.6 The Structure of Internet Software

Software from the Internet is structured in a way that is similar among packages. Most packages are large enough to comprise several files

which are typically transferred separately and simply appended to each other before being uncompressed. Compression techniques consist of the following:

1. compress/uncompress—This creates a file with a .Z extension.
2. pack/unpack—This creates a file with a .z extension.
3. gzip—This is a GNU compression algorithm; it also creates a file with a .z or .gz extension.

These algorithims are the three most common and should describe most files you find on the Internet. You typically save anywhere from 30 to 60 percent on file size using these algorithims depending on the file structure and size. The smaller the size, the less efficient the algorithms typically are. For example:

```
$ ls -l stuff*
-rwxr--r-- 1 kevin 61400 Jan 29 12:00 stuff.txt
$ compress stuff
$ ls -l stuff*
-rwxr--r-- 1 kevin 26100 Jan 29 12:00 stuff.txt.Z
```

Note that the compress algorithm renames the file appending a .Z to the original filename and saves approximately 55 percent file space. To uncompress the file, use:

```
$ uncompress stuff.txt
$ ls -l stuff*
-rwx-r--r-- 1 kevin 61400 Jan 29 12:01 stuff.txt
```

Note that the original file is restored and the input filename to use with the uncompress is the original filename without the extension. The pack algorithm works the same way and both pack and compress are equally utilized on the Internet. If you look on a remote system with anonymous FTP and see extensions like .Z or .z, you can be sure that these are compressed or packed files and should be transferred and undone at the local machine.

gzip is a GNU product which contains no copyrighted source code and is compatible with both pack and compress. There are options on the gzip command line which document how to handle both packed and compressed files as well as its own gzip format. Most, if not all, GNU software is now distributed in gzip format. See Secs. 6.7 and 7.1 for more information.

If the files come separated into named parts such as PART01, PART02, etc., you must append them together with a simple command like:

```
$ cp PART* >> newfile.tar.z
```

where newfile.tar.z is the name of the compressed file you can now uncompress. It is common that a file named README contains checksums of the PART* files. This allows you to check to see that the file transfer occurred correctly. Run either the sum or the cksum command on the PART* files and compare them against the files and associated numbers in the README file. One note of caution: There are two primary algorithms used to generate the checksum numbers and they will *not* produce the same result. If you run the checksum on several files and they are wildly different, it is a pretty safe bet that you have a sum or cksum command that is not the same as that used to compute the checksums in the README file.

The most common directory structure on an anonymous FTP machine is the pub or public directory. This typically contains all public domain software and is the first place you should look for software. The other common directory is the gnu or GNU directory. This contains software from GNU which is free as well. See Sec. 6.7 for more information. At the top of the pub directory, there is typically a file called ls-lR or index on an anonymous FTP machine. You should examine this to understand what is on that particular machine and choose what you would like to transfer based on what you learn.

6.6.1 Tar file archives

Often, entire directory structures are stored in tar files and then compressed using the pack or compress algorithm. This example illustrates the process of getting such a file and restoring it to a local directory:

```
$ ftp dopey
Connected to dopey
220 dopey FTP servers awaits your command
Name: anonymous
230 Guest login okay, send ident as password
password:
331 Guest login ok, access restrictions apply
ftp> binary
ftp> cd pub
ftp> ls
200 PORT command successful.
150 Opening ASCII mode data connection for /bin/ls.
total 4096
drwxr-s-r-x 2 root 120 512 Nov 10 12:05 dir1
-rw-rw-r-- 1 lib 120 47400 Jan 20 05:45 package1.tar.Z
-rw-rw-r-- 1 lib 120 32300 Jan 18 09:00 package2.Z
226 transfer compete.
220 bytes received in 0.1 seconds.
ftp> get package1
200 PORT command successful.
150 Opening BINARY mode data connection for package1.tar.Z (47400
bytes)
```

```
226 Transfer complete.
47400 bytes received in 1.0 seconds (47 kbs)
ftp> quit
221 Goodbye.
```

You must now uncompress the file and tar it into its proper form:

```
$ uncompress package1.tar
$ tar -xvf package1.tar
$ ls -l
-rw-r--r-- 1 kevin 4600 Sep 1 README
-rw-r--r-- 1 kevin 50001 Aug 20 package1.c
-rwxr--r-- 1 kevin 59000 Aug 20 package1
```

You may want to use the tar -tvf command as well to see the table of contents of the tar file before unwinding it since things such as pathnames and file sizes may determine how and where you untar the file.

Inside tar files you will typically see a file named README or READ.ME or some other variation. This should be the first file you look at after unwinding the tar file. In fact, you should extract the README file separately before unwinding the entire tar file to examine it for any issues relating to disk space, installation procedures, etc. There is also a Makefile which contains all necessary commands and information to build the product on a particular machine. Frequently there is also an Imake file which allows you to make a Makefile for a particular machine. You should examine the tar file for these before using the product since you may have to rebuild the product before execution will occur correctly.

As mentioned earlier, you may also see compressed files broken into parts to allow for easier transmission. Each piece is typically 100KB in size and numbered sequentially beginning wth 01 as an extension. For example, if package1 consisted of three pieces—package1.tar.Z.01, package1.tar.Z.02, and package1.tar.Z.03—you would bring over each file separately and put them back together using the cat command and its append capabilities. For example:

```
$ cat package1.tar.Z.* > package1.tar.Z
```

would place all files back into a single file named package1.tar.Z assuming the files were numbered sequentially. Then you would proceed as above.

The other common file naming convention for Internet files is partnn, where nn is an integer between 01 and some number. To generate a combined file from these files, use a command like:

```
$ cat part* > newfile
```

where newfile is the name of the file you would like to generate.

6.6.2 Shell archives

Shell archives consist of files which are merely shell scripts which unwind themselves by invoking themselves. To unwind a file called kevin.shar:

```
$ sh kevin.shar
```

You may see this divided as before, and you should apply previously discussed techniques before invoking the shell to unwind this archive. Be careful with these since they have been known to contain Trojan horses. You should examine each shell archive before unwinding it. As a general rule, you should also do all work related to the Internet as a nonroot account to ensure local system integrity.

6.6.3 Manual pages

Most software packages come with one or more manual pages to describe the usage of the tool. You can typically find the manual pages by looking in the main source directory for a file that has a single-digit file extension such as .1 or .2. A quick and dirty command to look for these files is:

```
$ ls *.?
```

which tells AIX to look for a file with a single character file extension. If you see anything with a numeric extension (1–8), it is probably a man page. Man pages are written in nroff format, which is a command-driven text processing language.

Everytime you invoke the man command, man searches for a matching man page in a certain area on the filesystem, and if it finds it, it executes an nroff command on the file. This displays the man pages in a formatted way on standard output (which is most often your terminal screen). You can issue this nroff command manually on a man page file. For example:

```
$ nroff -man gcc.1 | more
```

will print out the manual page for the GNU C compiler on your screen and pipe the output to more so you can page through at your own speed. With the an macros (note that with the -m switch they spell man . . . clever, huh?), you can format almost any manual page for preview on your screen.

The other nice thing about this is that you can print out the man page by simply redirecting standard output. For example:

```
$ nroff -man gcc.1 | lpr
```

By using a pipe to the printer system, you will see the formatted manual page on your default printer. This is one of the first things you should do when you begin looking at and using a new product, whether it is from the Internet or already installed on the system you are using.

6.7 GNU and Their Paradigm

GNU stands for GNU's Not UNIX and is the brainchild of a man named Richard Stallman. While at M.I.T., Mr. Stallman wrote much of emacs and many other tools which are widely used in the UNIX community today. After getting frustrated with organizational and legal barriers to creating free software, he left M.I.T. and formed GNU. Today, GNU and the associated Free Software Foundation (FSF) are clearly the leaders in providing high-quality free software on the Internet. The FSF provides for the distribution of code and maintenance and some support of the GNU software. They can be contacted at:

Free Software Foundation
675 Massachusetts Avenue
Cambridge, MA 02139

They are interested in any support they can receive, particularly with respect to free hardware, labor, and money. This helps them in their effort to continue to develop and distribute quality free software.

Originally, GNU was a project to create a replacement for the UNIX operating system and environment. In doing this, hundreds of tools have been written ranging from text processing systems to editors and compilers. Most are excellent and none are poor. Mr. Stallman is seen as somewhat of a radical, particularly by those who want to sell software for money. There is an article in the GNU sections on most Internet machines called the GNU Manifesto. You should read this if you are interested in getting a better understanding of Mr. Stallman's philosophy and, subsequently, of GNU's philosophy on software (it is included on the CD with this book and in App. C). It doesn't take long to understand that Mr Stallman has some pretty radical ideas about intellectual property and software. This philosophy has driven him to develop and distribute free software on the Internet.

The quality of the GNU software is high and the support and maintenance is good. There is much fear, uncertainty and doubt (FUD) about free software, most created by software vendors who sell products. They claim that support and quality of free products is low. This is simply not true. emacs is one of the best editors, if not the best, available today on every platform it runs on. Keep this in mind when you are thinking about using free software in your environment. Free soft-

ware, particularly GNU software, is of excellent quality, and you will have very little trouble with these products.

A few of the more popular GNU utilities are outlined in this book; however, there are many more available. See Sec. 6.9 for more information on how to get listings of GNU and other free software from the Internet.

GNU and the FSF don't have copyrights on their software; instead they have what they call a copyleft. This provides that the software can be distributed for a fee; however, it also says that anyone who distributes the software cannot stop it from being distributed again for whatever cost and in whatever manner anyone sees fit. This fits in with their philosophy of freely sharing software and information in the hopes that this will only improve the product and make it accessible to everyone who wants it. A copy of the GNU Manifesto, their COPYING information, and the GNU General Public License is included on the CD with this book as well as in App. C. These documents are included with all GNU software in this book and must be included in all distributions of their software.

6.8 How to Locate and Retrieve Software from the Internet

For years, the location of software on the Internet was communicated through information networks of people who were "in the know." Because of the tremendous growth of the Internet in the last few years, this methodology is no longer sufficient to allow all Internet users to know where all the software they are interested in exists. Because of this, several new systems have been developed to support search and retrieval capabilities from this massive network.

6.8.1 Archie

Archie was developed by several people at McGill University in an effort to consolidate file information on a few servers located around the world. By periodically polling servers which are contained in a list which the Archie people maintain, they create a list which can be accessed and searched with either telnet or mail. This list is an index of all files on those polled servers with additional information which describes the contents of the files.

In other words, Archie provides an index to files on the Internet. This is a wonderful feature when you are looking for particular products or tools and are not sure of their location. With this tool, you can find files virtually anywhere on the Internet. Once you have found them, you can use one of the tools described in this chapter to retrieve them.

There are several Archie servers around the world including the following:

Server name	Geographical area covered
archie.ans.net	Sites connected to ANS access provider
archie.au	Australia and the Pacific Basin
archie.doc.ic.ac.uk	United Kingdom
archie.funet.fi	Europe
archie.ngam.ca	Canada
archie.rutgers.edu	Northeastern United States
archie.sura.net	Southeastern United States
archie.unl.edu	Western United States

Other servers are listed in the archive help file shown later in the chapter. Note that you should try to use the server closest to your geographical area since this will minimize the network traffic as well as the load on the Archie server.

telnet support for Archie. If you have IP and telnet capabilites to the Internet, you can use telnet to access the Archie servers. For example:

```
$ telnet archie.mcgill.ca
...
Connected to archie.mcgill.ca
Escape character is '^]'.
SunOS UNIX (services.bunyip.com)
login: archie
# Bunyip Information Systems, Inc., 1993, 1994, 1995

Welcome to the ARCHIE server

archie>
```

There are a variety of commands within Archie to search the database and print results:

`help`	Prints out help menus.
`list`	Lists anonymous FTP servers used by Archie.
`maxhits number`	Sets the maximum number of matches allowed.
`prog string`	String is the string to match against the files in the database; probably a filename or substring within a filename.
`servers`	Lists all current Archie servers.
`set variable`	Sets a boolean variable; use show to see possibilities.
`show variable`	Shows boolean variable. If you don't use variable, it will list all variables.
`set search type`	Type can be: exact—exact string match. regex—string is regular expression. sub—string is a substring, case independent. subcase—string is a substring, case must match.

`sho search`	Shows search type.
`unset variable`	Unsets a boolean variable.
`whatis string`	Matches string against index entries in database.

The most common way to use Archie is to search by filename. All products will contain files which are named in a way that is related to the product name. This allows you to search filenames collected by the Archie server by a string using the prog command:

```
ARCHIE> set search exact
ARCHIE> prog proton
proton: nothing appropriate

no matches
```

You can use the whatis command to search the associated index database. When software is added to the Archie database, the author can include information which Archie uses to index the files. This allows you to search for information that may or may not be included in the filenames for the product. This is a nice way to narrow a search; however, the information is often out of date and should be used cautiously. Once you get this information, you can access the server referenced with anonymous FTP as described earlier in this chapter.

Mail support for Archie. If you do not have full IP access to the Internet, you will not have telnet access to the Archie servers; however, you still have mail support. Support for Archie available through the mail interface is very similar to that offered by telnet, and you will not suffer much loss of functionality. You may wish to submit a mail message to an Archie server and view the results the next day or simply do not have time or the inclination to peruse the Archie databases in real time. Many people who have full telnet access to the Internet still use the mail interface to access and search the Archie servers.

To access the Archie server via e-mail, simply address the mail to:

```
archie@server
```

where server is one of the servers listed earlier in this chapter.

Archie supports what is known as a mail server. This is a process which runs as a mail background process which, when invoked, processes incoming mail without human intervention. Many organizations on the Internet use this to serve files and information via e-mail. Often, the command to the mail server is contained in the subject line. Archie requires no subject line information but instead expects the commands in the text of the message beginning in the first column. You can simply build commands in the text field of the mail message beginning in the

first column. Any command that is not recognized is treated as a help command, and you will get a help file returned for that command in the mail message.

Commands supported by the mail server Archie servers are a subset of the telnet Archie commands, which are:

`compress`	Causes returned mail to be compressed and uuencoded before being sent
`help`	Causes a help guide to be returned
`list expression`	Provides a return list of servers matching expression
`path address`	Gives return e-mail a path other than that contained in the from: field
`prog expression`	Searches for filenames matching the regular expression
`servers`	Causes a list of Archie servers to be returned
`whatis string`	Causes a list of possible files with string matched in the index database
`quit`	Ends session

To get a help file on using Archie via e-mail, issue the commands:

```
$ mail archie@archie.rutgers.edu
Subject:
help
.
```

Sometime later you will receive a file containing help information on Archie and supported commands. For example:

```
From archie-errors@dorm.rutgers.edu Mon May 29 23:16:36 1995
To: kleining@devtech.devtech.com
From: (Archie Server)archie-errors@dorm.rutgers.edu
Date: Mon, 29 May 95 23:10 -0400
Subject: archie [help] part 1 of 1
> path kleining@devtech.devtech.com
> help
        Archie Email Help (Version 3.2)
HELP for this archie email server, as of 11 April, 1994.
To perform an archie search via email, send mail to
        archie@archie_server
where <archie_server> is the name of an archie host, some of which
are listed below.
The "Subject:" header in mail sent to archie is treated as part of
the message body.
Command lines begin in the first column. All lines that do not match
a valid commands are ignored.
Empty messages are treated as "help" requests (this file). If no
command in a particular message can be recognized, the message is
treated as "empty" and this file will be returned.
The current (and complete) list of archie servers can be found with
the "servers" command (described below). A sample list is:
    archie.au                   139.130.4.6          Australia
    archie.edvz.uni-linz.ac.at  140.78.3.8           Austria
    archie.univie.ac.at         131.130.1.23         Austria
    archie.uqam.ca              132.208.250.10       Canada
```

```
    archie.funet.fi             128.214.6.102       Finland
    archie.univ-rennes1.fr      129.20.128.38       France
    archie.th-darmstadt.de      130.83.128.118      Germany
    archie.ac.il                132.65.16.18        Israel
    archie.unipi.it             131.114.21.10       Italy
    archie.wide.ad.jp           133.4.3.6           Japan
    archie.hama.nm.kr           128.134.1.1         Korea
    archie.sogang.ac.kr         163.239.1.11        Korea
    archie.uninett.no           128.39.2.20         Norway
    archie.rediris.es           130.206.1.2         Spain
    archie.luth.se              130.240.12.30       Sweden
    archie.switch.ch            130.59.1.40         Switzerland
    archie.nctuccca.edu.tw                          Taiwan
    archie.ncu.edu.tw           192.83.166.12       Taiwan
    archie.doc.ic.ac.uk         146.169.11.3        United Kingdom
    archie.hensa.ac.uk          129.12.21.25        United Kingdom
    archie.unl.edu              129.93.1.14         USA (NE)
    archie.internic.net         198.49.45.10        USA (NJ)
    archie.rutgers.edu          128.6.18.15         USA (NJ)
    archie.ans.net              147.225.1.10        USA (NY)
    archie.sura.net             128.167.254.179     USA (MD)
If you do not get mail back within 1 day or so, try using the "path"
command described below.
Mail destined for the ADMINISTRATION of individual servers should be
addressed to:
       archie-admin@archie_server
where <archie_server> is one of the hosts listed above. If you are
having a problem with a particular server, try sending mail to its
administrator first before contacting the general archie contact
address below. They may already be aware of the problem.
To request the ADDITION or DELETION of a site from the archie
database, send mail to:
       archie-admin@bunyip.com
To contact the IMPLEMENTORS of archie, send mail to:
       archie-group@bunyip.com
For your information anonymous FTP may be performed through the mail
by various ftp-mail servers. Send a message with the word 'help' in
it to:
For BITNET/EARN sites ONLY:
       bitftp@pucc.princeton.edu
or (general access):
       ftpmail@decwrl.dec.com
for an explanations on how to use them.
Under version 3.2 the email client implements all the
non-interactive commands and variables of the telnet client.
However, interactive commands like "pager" are not supported as they
don't make much sense in the email environment.
For a complete explanation of the the archie system use the
"manpage" command to request a copy of the manual page, what follows
is a short summary of the valid email commands and variables.
NOTE: The "site" command of earlier versions of archie has been
disabled under version 3.2 until it can be reimplemented with the
new architecture of the system.
"Quick and dirty" summary
-----------------------
For those of you who want to get something done now and read the
rest of this later, send the email to an archie server with the line:
       find <foo
(where <foo> is the name of the file you are looking for). You
should get a message back with results of your search. If you want
to be a bit more sophisticated, read on. . . .
```

```
Commands
--------
In the commands that follow, parameters between '[' and ']' are
optional. The ellipsis ("...") signifies that the previous parameter
can be repeated multiple times. A '|' character means "or".
help [ <topic> [[ <subtopic> ] ...]]
                The "help" command by itself produces this message.
                An optional topic and subtopic(s) may also be given.
                A list of words is considered to be one topic, not a
                list of individual topics. Thus,
                     help set maxhits
                requests help on the subtopic 'maxhits' of topic
                'set', not on two separate topics.
find <pattern   This command produces a list of files matching the
                pattern <pattern. The <pattern> may be interpreted
                as a simple substring, a case sensitive substring,
                an exact string or a regular expression, depending
                on the value of the variable search.
prog <pattern   This is identical to "find" and is included for
                backward compatibility with older versions of the
                system.
list [ <pattern> ]
                Produce a list of sites whose contents are contained
                in the archie database. With no argument all the
                sites are listed. If given, the <pattern> argument
                is interpreted as a regular expression (see the
                archie manual page for an explanation of regular
                expressons) against which to match site names: only
                those names matching are printed. The format of the
                output can be selected through the output_format
                variable (described below).
                Note that the numerical (IP) address associated with
                a site name is valid at the time the site was last
                updated in the archie database, but may have changed
                subsequently.
mail <address   Mail the results generated up until this command to
                <address. This must be a vaid email address.
manpage [ roff | ascii ]
                Return the archie manual page. The optional
                arguments specify the format of the returned
                document. "roff" specifies UNIX troff (or nroff)
                format, while ascii specifies plain, preformatted
                ASCII output. With no arguments it defaults to ascii.
motd            Re-display the "message of the day", which is
                normally printed at the start of the returned
                message.
path <address   Set the return email address to <address. This
                overrides the default path which the system
                automatically generates by looking at the incoming
                mail header. This is actually an alias for "set
                mailto <address" (see "Variables" below), and is
                included for backwards compatability.
servers         Display a list of all publicly accessible archie
                servers worldwide. The names of the hosts, their IP
                addresses and geographical locations are listed. IP
                addresses were valid at the time that this document
                was last updated.
domains         Give a list of the archie pseudo-domains that the
                archie server supports. See the manual page for an
                explanation of archie pseudo-domains.
set <variable> <value
                Set the specified <variable> to <value. See
                "Variables" below.
```

```
show [ <variable-name...]
                Without any parameters, display the status of all
                the user-settable variables, including such
                information as its type (boolean, numeric, string),
                whether or not it is set and its current value (if
                its type requires a value). Otherwise show the
                status of each of the specified arguments. Useful
                for finding out what the default settings at a
                server are.
unset <variable Unset the specified <variable. The subsequent value
                of the variable is defined on a <variable-specific
                basis.
version         Print the current version of the email interface.
whatis <substring
                Search the Software Description Database for the
                given substring, ignoring case.  This database
                consists of names and short descriptions of many
                software packages, documents (like RFCs and
                educational material), and data files stored on the
                Internet.
                Note that this database is currently maintained by
                hand and is certain to be outdated (the net changes
                on a daily basis).
Variables
---------
The archie email system has 3 types of variables.
1) Numeric
----------
Numeric variables may have preset internal ranges in which the value
of the variable must lie.
maxhits         Allow the "find" command to generate at most the
                specified number of matches (hits) (permissible
                range: 0-1000). Default 100.
maxhitspm       Maximum number of files (hits) per filename located
                in the find command. See the manual page for more
                information. Range 0-1000. Default 100.
maxmatch        Maximum number of filenames to return with the find
                command. This is NOT the same as maxhits which
                limits the total number of files returned. See the
                manual page for more information. Range 0-1000.
                Default 100.
max_split_size  Approximate maximum size, in bytes, of a file to be
                mailed to the user. Any output larger than this
                limit will be split in pieces of about this size.
                This can be set by the user in the range 1024 to 2Gb
                with a default of 51200 bytes. Some mail gateways
                will not allow results of over 100Kb and so care
                should be taken when setting this limit.
2) String
---------
String variables may have a predefined range of values.
compress        The kind of data compression the user can specify
                when mailing back output.  Currently allowed values
                are "none" and "compress" (standard UNIX compress
                program) with a default of "none"
encode          The type of post-compression encoding the user can
                specify when mailing back output.  Currently allowed
                values are "none" and "uuencode", with a default of
                "none". Note that this variable is ignored unless
                compression is enabled (via the compress variable).
language        Allows the user to specify the language in which the
                help, etc. is presented. Individual servers may be
                configured for a range of languages.
```

```
mailto          A valid address to mail the results back to. This
                overrides the address automatically generated by the
                archie system from the incoming mail header. Setting
                this variable is equivalent to using the "path"
                command.
match_domain    Restrict the returned files to sites in the colon-
                separated list of domains and pseudo-domains. See
                the manual page for further information.
match_path      Restrict the files returned in the 'find' command to
                contain the colon-separated list of pathname
                components. See the manual page for further
                information.
output_format   Affects the way the output of "find" and "list" is
                displayed.  User settable, with valid values of
                "machine" (machine readable format), "terse" and
                "verbose", with a default of "verbose".
search          The type of search done by the "find" (or "prog")
                command. The list of valid values is given below in
                order of increasing search times. The given search
                string may match a directory or filename in the
                database.
                exact           String has to match exactly
                                (including case)
                subcase         Substring match. Case sensitive.
                sub             Substring match. Case insensitive.
                regex           Regular expression (see ed(1)) search.
                                Case sensitive. The archie manual
                                page gives examples of regular
                                expressions.
                There are also compound searches made up of
                combinations of the above search methods in sequence:
                exact_sub       Try "exact". If no matches found use
                                "sub".
                exact_subcase   Try "exact". If no matches found use
                                "subcase"
                exact_regex     Try "exact". If no matches found use
                                regex.
                Note: unless specifically anchored to the beginning
                (with ^) or end (with $) of a line, regular
                expressions (effectively) have ``.*'' prepended and
                appended to them. For example, it is not necessary
                to type
                     find .*xnlock.*
                because
                     find xnlock
                In this instance, the regex match is equivalent a
                simple substring match which should be used instead.
server          The archie/Prospero server to which the email
                interface connects when "find" or "list" commands
                are used. Usually defaults to "localhost" on most
                archie systems.
sortby          Set the method of sorting to be applied to output from the
                "find" command.
                The five permitted methods (and their associated
                reverse orders) are:
                none            Unsorted (default; no reverse order,
                                though 'rnone' is accepted)
                filename        Sort files/directories by name, using
                lexical order (reverse order: 'rfilename')
                hostname        Sort on the archive hostname, in
                                lexical order (reverse order:
                                'rhostname')
```

```
                size            Sort by size, largest files/
                                directories first (reverse order:
                                'rsize')
                time            Sort by modification time, with the
                                most recent file/directory names
                                first (reverse order: 'rtime')
3) Boolean
----------
Currently the email interface to the archie system has no variables
of type "boolean".
```

This is an exact duplicate of the message I received when I issued the help command to an Archie server. You should get something similar. As an example, if you want to find all information related to protons, you would do something like:

```
$ mail archie@archie.rutgers.edu
Subject:
whatis proton
.
```

You would receive information with all files found with the corresponding index keyword of proton. For example:

```
From archie-errors@dorm.rutgers.edu Mon May 29 23:16:35 1995
To: kleining@devtech.devtech.com
From: (Archie Server)archie-errors@dorm.rutgers.edu
Date: Mon, 29 May 95 23:10 -0400
Subject: archie [whatis proton] part 1 of 1
> path kleining@devtech.devtech.com
> whatis proton
# Nothing found that matched 'proton'.
```

In the above example, there were no matches found. An example of something with a match might look like:

```
$ mail archie@archie.rutgers.edu
Subject:
whatis gcc
.
```

You could then search for information on a particular file within the whatis returned list:

```
$ mail archie@archie.rutgers.edu
Subject:
prog gcc
.
```

You now have information describing the location and contents of an archive on the Internet. You can retrieve it based on FTP or another method supported by your Internet access provider.

6.8.2 FTP servers

As mentioned earlier in this chapter, you can use mail to access FTP servers if you do not have IP access to the Internet. To do this, you issue a set of commands which essentially build a script to be executed on a remote mail server. The basic set of commands available within the mail message is:

`binary`	Specifies that the file is binary and needs to be encoded with btoa before transmission begins
`chdir dir`	Specifies a directory to cd to
`chunksize size`	Specifies a maximum file size for transmission
`compress`	Uses compress to compress the file before transmission
`connect [hostname [login [passwd]]]`	Specifies hostname to connect to and account to use
`dir [dir]`	Generates a directory listing of the current or specified directory
`get file`	Specifies the file to be sent to you via e-mail
`quit`	Ends the request
`uuencode`	Specifies that the file be uuencoded before transmission instead of btoa

The FTP mail server will follow all commands as you place them in the mail message. The Subject: field is ignored, so begin your script at the beginning of the mail text. The result of the query will be mailed to you electronically. It may take some time for the request to be serviced (a matter of hours) since the FTP mail server that you may use may be heavily loaded with requests or other tasks, and it may take some time before it gets to your request. The help file tells you where to send mail if you have trouble.

The only other thing to note about the FTP mail server is that it breaks files into 64,000-byte files. If the file to be transmitted is larger than 64,000 bytes, the FTP mail server will break it into as many 64,000-byte chunks as necessary and will transmit these files one by one. You can change this chunksize with the chunksize command as described above. Note also that you have to tell the FTP mail server that you are transferring a binary file if you don't want any translation to occur.

A simple example of this is:

```
$ mail ftpmail@decwrl.dec.com
Subject:
ls
quit
```

This will list all files in the working directory on the default machine decwrl.dec.com. Note that the userid is anonymous unless specified otherwise.

A slightly more sophisticated example is:

```
$ mail ftpmail@decwrl.dec.com
Subject: anything goes
connect uunet.uu.net
chdir /index/networking
get by-name.Z
ls
quit
```

This will retrieve a file named by-name.Z on the UUnet machine in the directory /index/networking. It will also generate a listing of the directory contents after the file inclusion. All of this is in the mail message you will receive back from the FTP mail server machine. The other point to note is that you can only issue the chdir command once within a FTP mail script. Therefore, if you want to get more than one file from different directories, you must send separate mail messages.

Be careful not to confuse the FTP mail server syntax with Archie syntax since they are clearly not the same. Keep this in mind if you get some confusing error messages back from either server.

6.8.3 Gopher

Gopher is a system much like Archie in that it gives you the ability to search for topics and files on the Internet at random. Gopher was written at the University of Minnesota and consists of servers storing information on virtually any topic. Along with the Gopher servers, you need a Gopher client to access all information on the Gopher servers. This service provides you with a menu-driven interface to a variety of information servers. It is not necessary for you to know where the information is you are looking at or in what format it is stored, only that you can access it if you can see the menu item in Gopher.

Gopher clients come from the Internet in a variety of formats, but the one you will probably be interested in is the X Windows gopher client. This is available on the boombox.micro.umn.edu machine in the pub/gopher directory. Use FTP or simply telnet to this machine and download the Gopher client as you desire. The Gopher client comes hard-coded with a Gopher server name and address in it, and this will be a good first connection once you bring up the Gopher server.

There are two Gopher public servers from which you can try Gopher before installing it on your machine: consultant.micro.umn.edu and gopher.uiuc.edu. You can simply telnet to these machines, log on, and invoke Gopher to see the standard curses AIX interface to Gopher. Make sure you have defined your terminal type correctly, or you may have some strange screen behavior.

The Gopher server knows the format of any resource available to you

on the network and formats the appropriate command based on type of resource you are interested in. As you continue down into the Gopher menus, you will eventually reach an actual file or piece of information which requires some action. If it is a file, Gopher will automatically invoke an FTP session for you and will transfer the file. If it is a login resource, Gopher automatically creates a telnet session and so on.

A typical first Gopher screen looks like:

```
$ gopher
Internet Gopher Information Client v0.9
                Root Directory

--->   1. Welcome to the U of Illinois Gopher
       2. CCSO Documentation/
       3. Computer Reference Manuals/
       4. Frequently Asked Questions/
       5. GUIDE to the U of Illinois/
       6. Libraries/
       7. National Weather Service/
       8. Other Gopher and Information Servers/
       9. Peruse FTP Sites/
       10. Phone Books/

       Press ? for Help, q to Quit, u to go up Page: 1/1
```

The / at the end of some lines denotes that this is a directory or submenu, while the other lines represent resources. By moving the cursor to the appropriate line, you can control the context of your environment. Note that based on the lines above, you can access many types of information from this one menu. Note also that option 8 provides you with the capability to move to other Gopher servers and sites.

This is a very brief overview of some of the capabilities and functions of Gopher. It is an extremely powerful tool to help you to search and use the Internet more effectively.

Gopher is not the only tool to perform this searching, however. There are other, newer tools which provide significantly enhanced functionality, such as those documented below.

6.8.4 WAIS

WAIS stands for Wide-Area Information Servers. The WAIS system consists of a variety of WAIS database servers which provide text search capabilities to anyone running a WAIS client. WAIS is based on an ANSI draft standard known as Z39.50 which has been under development for some time in the library community. Z39.50 specifies an architecture consisting of information servers which provide text search services to clients. WAIS clients are available in a variety of formats including X Windows and curses. To get the WAIS client, FTP from

think.com in the directory WAIS. There are a variety of clients based on machine type from which you can download a WAIS client.

When you issue a search request from a WAIS client, it queries a list of servers for possible matches. Each server then sends a matching listing back to the client, and the client normalizes the response matches and displays them. The system with the most relevant matches has a value of 1000, while the others have proportionally less. This allows you to move immediately to the document and system which is most likely to be a good match.

This system is not flawless by any stretch and contains many strange characteristics and quirks. However, it does provide a relatively simply querying system for the Internet.

There are several example WAIS clients on the network including quake.think.com and nnsc.nsf.net. You simply log in as WAIS and proceed from there. This is a good way to see how a real live WAIS client works before you download your WAIS client.

WAIS has a relatively stupid interface and will not understand complex queries and relational operations such as OR or AND. You must simply specify words you would like to match as a query and then issue the query to the known list of WAIS servers. Each WAIS client contains a listing of known WAIS servers, and from this you can generate a larger list of servers if you so desire.

The best way to learn more about WAIS is to log on to one of the example WAIS client machines as mentioned above and try it for yourself. If you find it useful, simply download the appropriate client and have at it.

6.8.5 WWW

The final tool relevant to this book and Internet tools is the World Wide Web (WWW). This is a tool which provides hypertext capabilities for access to the Internet. As with the other tools there are multiple WWW clients including Mac, PC, and AIX. These support both character mode and a graphical interface to WWW.

WWW is based on the concept of a home page. The home page is the intial page of documentation made available to you when you first enter WWW. WWW organizes all information on the Internet into hypertext documents in which you can jump around based on the hypertext capabilities of WWW. An example home page is shown in Fig. 6.5. The numbers denote hypertext links. To move from one page to another, simply press the number next to the information you are interested in and press RETURN. This will automatically move you to this document. From here, you can again move through the document or use hypertext to move to another document from it.

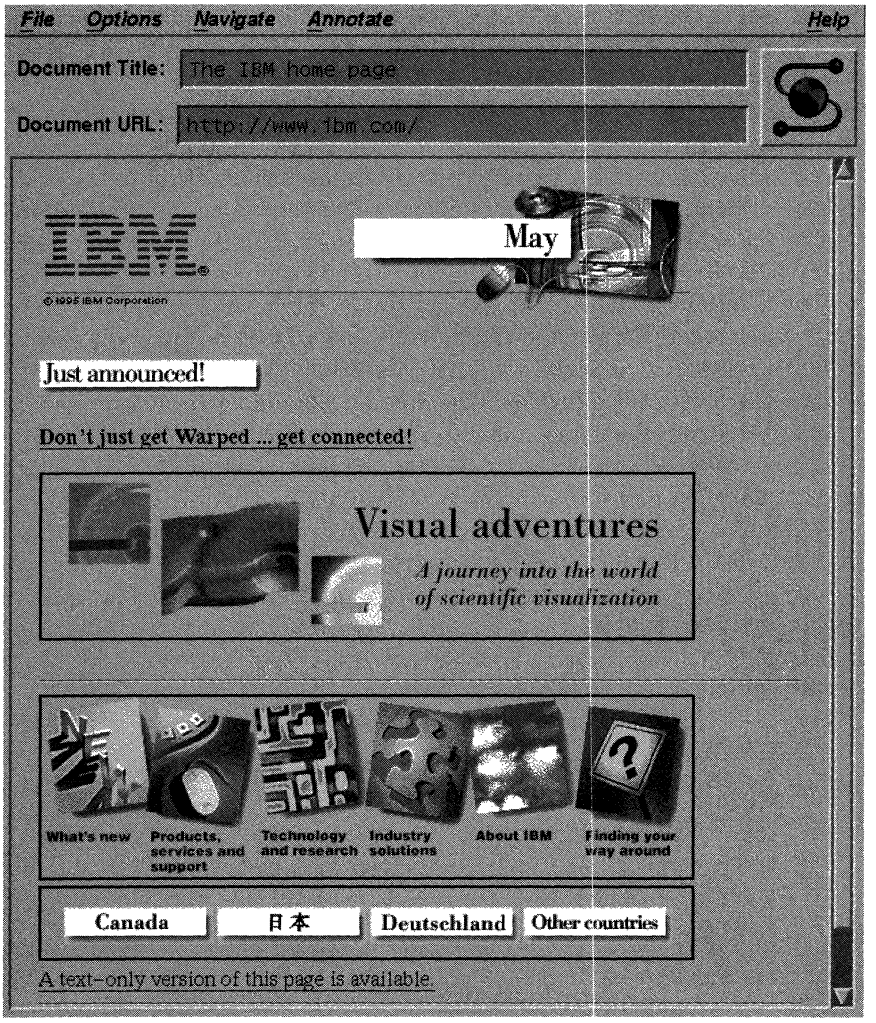

Figure 6.5 Mosaic.

There are several clients known as browsers available for WWW. One of the first was a browser called Viola. This is an XII-based WWW client application which provides graphical access to the Internet WWW. A more recent and powerful browser is known as Mosaic. This has significant potential and provides true multimedia capabilities to the average Internet user. Its potential successor is Netscape. Netscape is faster and more secure than Mosaic; however, Mosaic version 2.0 is now available and may prove to be the overall winner in the browser war.

WWW has not yet really been exploited in ways that make it truly revolutionary. However, in the next 2 to 3 years, you will see some dramatic changes in the way people use tools such as WWW to access the Internet, and this is going to cause a revolution in the way we live and work.

6.9 How to Build Software from the Internet

The following chapters on tools from the Internet contain sections on building the products discussed on the particular machine on which you are running. However, there is a commonality between all build procedures ranging from AIX machine to VAXes and mainframes.

Typically, an archive contains a build procedure for the type of machine it is intended to run on. For UNIX machines, they are typically makefiles which consist of commands to build all necessary files for the particular UNIX "flavor" you are running on. For other machines there are similar files which contain build procedures provided by the software developer to ensure that the product builds properly and runs correctly the first time. Since the focus of this book is AIX, the discussion of the formats and procedures is limited, but keep in mind that procedures for other non-UNIX platforms are similar based on the functionality of the underlying platform.

Traditionally, when you retrieved the software from the Internet, you had to look through the makefile and make modifications depending on what architecture you were running on. Different versions of UNIX have different C compilers, different library locations, and different versions of make, just to name a few. Consequently, it has sometimes been difficult to build software from the Internet for certain platforms.

GNU has made great strides in this area, and they are now providing a shell script called configure. Most GNU software distributions contain a file called INSTALL which details much of what you need to know regarding the installation and compilation of all software in the distribution. Now, instead of having to edit the makefile, you simply type:

```
./configure
```

from the main directory, and the configure script creates a correct makefile. Note that it assumes you have your software in the /usr/local directory and attempts to place executables in /usr/local/bin, libraries in /usr/local/lib, and man pages in /usr/local/man/man1. If you wish, and you probably should, to place the resulting files in some other directory, you need to specify a series of commands to the configure

and/or make commands so that the appropriate files will be generated in a subdirectory that is different from the default.

For all Internet tools described in this book, there is a discussion of how to build them, and this will show how to build a particular tool into recommended directory structures. See Sec. 7.4 for more information on the build process.

6.9.1 Distribution format

AIX archives are typically in one of two formats: tar or cpio. Both are standard AIX utilities which have their own behaviors and characteristics. The structure of software from the Internet was discussed in a previous section; suffice it to say that you "unwind" the file in whatever format you receive it in. Compression is also an issue, and you must ensure that you have unpacked or uncompressed the file appropriately before beginning installation. See Sec. 6.9 for more information on the build process.

Once the software has been restored, it must be built. There are several "standard" files with each software distribution. Typically, there is a README or README.1ST or something similar contained within the directory structure. This should be the first thing read before beginning to build any compilations. It contains release and other information pertinent to the building of the product.

6.10 Understanding Internet Software Documentation

Most Internet packages have software documentation distributed with them in the form of nroff man pages and/or texinfo documents. These tools are discussed in Secs. 9.22 and 8.3, respectively; however, it is worth mentioning here the basic structure of these files in a typical distribution.

Most tools come with manual pages which typically end in either .1 or .man. These are usually in nroff format and can be viewed with the command:

```
$ nroff -man file.man | more
```

This is what the man command does to unformatted man pages when you issue the man command itself. While you can view these files with the man command, you have to construct the proper "MANPATH" and the proper structure of the directories for the man command to find the proper input file. It is most often easier simply to issue the above command.

There is also often a texinfo file distributed with most Internet prod-

ucts. This file typically ends in .texinfo and consists of input formatted for the texinfo reader in emacs. This provides a pseudo-hyptertext-based system which allows you to jump between points in the document. You can also use TeX to process the texinfo file into printed output. This allows you to use one file for both screen display and paper output. You can also use the stand-alone texinfo reader included on the accompanying CD to view the texinfo documents even if you don't have emacs.

The final way you may see documentation is in Postscript. These files typically end in .PS or .ps. You can print these out or display them on your screen with tools like Ghostscript and Ghostview.

Sometimes there are doc and man directories and sometimes there aren't. This is up to the person constructing the distribution, and there are no hard and fast rules. Simply scan for these files and directories when you unwind the product.

Much of the information in this book is available in the documentation delivered with most of the products; you just have to know where to look. As you use Internet software and other AIX tools more frequently, you will learn how to find your way around the system.

6.11 FAQ

Related to the documentation of a tool are frequently asked questions (FAQ) files. These files contain questions and answers to frequently asked questions on a particular topic or tool. Several are included on the accompanying CD, ranging from X11 to Motif and OpenLook. There are other FAQ files, and you can peruse the Internet for more of these on your particular topic. The files may also be named faq or some combination of product name and faq or FAQ. Search for faq or FAQ to get more information about products on the Internet.

6.12 Internet Futures

The future of the Internet is unclear as of this writing. The only thing that is clear is that the fundamental nature of the Internet has changed forever in the last 12 to 18 months. With the explosion of commercial access utilizing new technologies such as the World Wide Web, Mosaic, Netscape, and others, the Internet is growing at rates never dreamed of by its early users.

Because of the explosion of growth on the Internet, IP addresses are fast becoming a precious commodity. The IETF is currently working on the next generation of the IP protocol known as IP version 6 (IPv6.) The current version of IP is IP version 4 which uses a 32-bit address space.

IPv6 is based on a 128-bit address, which will allow for the representation of significantly more addresses than currently supported.

Commercial business is jumping on the "information superhighway" bandwagon in a big way, and services such as on-line banking, on-line trading, on-line shopping, and virtually any other activity that you can imagine is or will soon be occurring on the Internet.

From a developer's perspective, the tools discussed in this book and many like them are still stored on anonymous FTP servers around the world. There is a particularly good AIX software server known as aixpdslib.seas.ucla.edu from which some of the software on the accompanying CD was retrieved.

Much is being written about the Internet, but its ability to provide high-quality software and discussions are its largest value to the software developer.

Chapter

7

Nonnative Software Development Tools

Development software includes software which has been and can be used to generate other software systems on AIX. While there are many tools included in this book which do not support the development process directly, they are useful in increasing the productivity of the software developer. However, these tools are discussed in a different section of the book. This chapter focuses on tools which are directly used to generate software systems.

Even though the XL C compiler system that comes with AIX 3.2.5 is very good, there are many reasons you may want to use a different C compiler technology such as GNU C. One reason is that AIX 4.1.*x* does not include a C compiler by default. Multiplatform support, high performance, cross-compilation capabilities, and other capabilities of the GNU C compiler system have led to a large following for this compiler. Because of this, it is discussed in the second section of this chapter. All programs on the accompanying CD as well as all examples in this book, unless explicitly noted, have been built with the GNU C compiler.

Another issue which must be addressed before you can begin to build Internet software such as GNU C is that the software is distributed in compressed (gzipped) format. gzip is the GNU compression/decompression system which supports not only pack and compress format files but its own format as well. This is why gzip is the first tool discussed in this chapter.

The chapters in the rest of the book outline software packages that are available from the Internet and a variety of other sources including IBM. Until recently you had to jump through hoops to get IBM to send you patches and fixes to support some of the software contained on the

accompanying CD. This is no longer true. With the availability of the FixDist server and the aispdslib server on the Internet, anyone can now get access to much of the information contained in this book and elsewhere.

Much of the software on the accompanying CD was retrieved from the machine named aixpdslib.seas.ucla.edu (128.97.2.211). This machine contains a variety of directories which contain FAQs, software source code, precompiled software systems and a variety of other information that is very valuable to the AIX user. Simply use the anonymous FTP capability of any machine connected to the Internet and access this wealth of information.

IBM now also has the fixes mentioned in this book for the AIX operating system including the PTFs available on the Internet. This service is called FixDist and stands for Fix Distribution. The best way to begin to use FixDist is to anonymous ftp to aix.boulder.ibm.com (198.17.57.66) and download the FixDist software. Once you have installed this on your AIX machine, you can access the FixDist server and download patches and other software related to AIX. This will prove to be invaluable to you as you continue with this book.

These are two of the most important resources available to you for AIX, and you will want to keep them in mind as you proceed with nonnative AIX software.

7.1 gzip

7.1.1 Introduction

gzip is the GNU compression algorithm used to compress most GNU software products for distribution. gzip supports both compression and uncompression of its own format and also supports the decompression of pack and compress files. This makes it the ideal tool for compressing and uncompressing files on a UNIX platform. Note that gzip and the often-used PC program called zip are in no way related to each other.

The history of the gzip product is that the pack and compress routines that ship with UNIX contain an algorithm which is proprietary in nature. Because of this, it is necessary to pay royalties to the owner when these tools are acquired. As is often the case with Internet software, someone decided he or she didn't want to pay royalties to these people and created his or her own algorithm for compression and decompression.

It turns out that the compression algorithm for pack and compress is proprietary, but the decompression algorithm is not. Therefore, gzip supports the decompression of both pack and compress files as well as its own gzip format. This means that there is virtually no file from the Internet that gzip cannot unwind. This makes it a very powerful tool

for use with Internet tools. Gzip can also unpack zip files (of DOS and Windows fame) assuming that they have only a single member and were compressed in the deflation method.

gzip contains associated tools for viewing compressed files without uncompressing, changing compressed files from one format to another (pack to gzip, etc.), and comparing compressed files without performing uncompression. These tools make it much easier to work with compressed files, especially when disk space is limited.

Since gzip is a GNU product, it is fully covered by the GNU General Public License and this is how it is distributed.

7.1.2 Usage

The basic syntax of the gzip command is:

```
gzip [-a][-c ][-d] [-f] [-h] [-l] [-L] [-r] [-S string] [-t] [-v]
[-V] [-#] [file ...]
```

where -a converts end-of-line using local conventions. This is useful for DOS to UNIX transfers.
-c writes output to standard output.
-d decompresses.
-f forces compression or decompression.
-h displays help.
-l lists file sizes before and after compression.
-L displays gzip license.
-n—when compressing does not save original filename or timestamp. When uncompressing, does not restore the original filename and timestamp.
-q is quiet mode.
-r follows all directories and uncompresses or compresses files in subdirectories as well as in the current directory.
-S string uses string as the file suffix instead of the default .gz.
-t checks compressed file integrity.
-v is verbose mode.
-V displays gzip version.
-# is number 1 through 9 for compression levels; 1 performs the compression the fastest but does the worst job, while 9 takes the longest but maximizes the compression. The default is 5.
file ... is one or more files to compress or decompress.

You can place gzip default options in the environmental variable GZIP. These will be overridden by any command line options that conflict with those in GZIP.

gzip reduces the size of files anywhere from 50 to 70 percent depend-

ing on the structure and size of the file. Small files are not as effectively compressed due to the efficiency of the compression algorithm.

When you use gzip to compress a file, it is replaced with a file of the same name followed by a .gz. When you uncompress a file, a file with a .gz extension is searched for and replaced with an uncompressed version without the .gz extension. Note that gzip recognizes the file extensions .tgz and .tax as .tar.gz and .tar.z, respectively, and renames the files accordingly.

Another command you can use is the gunzip command. The basic synax is essentially the same as for gzip. In fact, they are symbolic links to each other. The syntax is:

```
gunzip [-cdfhlLnqrtvV] [-S string] [file ...]
```

where the options are exactly the same as for gzip except for the -d, which is the uncompress option. gunzip is exactly the same as gzip -d.

You can also view files without actually uncompressing them with the command:

```
zcat [-fhLV] [file ...]
```

where the options are again the same as above. zcat will perform a cat (listing) of the information by uncompressing and piping the results to standard output, which is most often the screen. This is useful when you want to see what is contained in the file before uncompressing it, especially when disk space is limited.

You would typically pipe the output of the zcat command to a pager such as more or pg. Because of the frequency of this need, there is the zmore tool. The syntax is:

```
zmore [file...]
```

zmore uses its own pager but does consult the PAGER environmental variable if you want to define a different pager such as pg or less. Its basic default pager behavior is that of more. Some of the more basic commands available with this pager are:

=	Displays the current line number.
/string	Finds the next occurrence of string and positions this on top of the screen.
!command	Invokes a shell command and returns to current location.
.	Repeats previous command.
i-space	i is a number which represents the number of lines to display. A space bar by itself displays a page defined by the TERM environmental variable.
iz	i lines is the new default window size.
is	Skips i lines and displays a screenful.

`if`	Skips i screenfuls and displays a screenful.
`q`	Quits.

These are fairly standard pager commands and provide much of the functionality you will need when previewing a file. If you need more, use an editor.

gzip and related commands will automatically detect the structure of the compressed file and generate the correct action to uncompress the file. pack and compress files are automatically recognized and processed correctly, as is gzip's own format for compressed files.

Keep in mind that gzip understands compressed files with a .z extension, which is the same as the pack command. You may use gzip to work with packed files but not the reverse. You will discover this if you attempt to run the unpack command on a gzip format file.

If you want to compare compressed files, use the commands:

```
zcmp [coptions] file [file...]
zdiff [doptions] file [file...]
```

where coptions are options defined by your implementation of cmp.
doptions are options defined by your implementation of diff.
file—if only one file is specified, it is compared against a matching .z file; if more than one file is specified, it is compared with previous files.

zcmp and zdiff invoke cmp and diff, respectively. All zcmp and zdiff do is uncompress files and pass them to the cmp and diff utilities.

You can control the default behavior of gzip through the use of an environmental variable GZIP. You can define the default options for gzip by setting these in the GZIP variable. For example:

```
$ export GZIP="-r -v -9"
```

When you next invoke gzip, it will work with recursive and verbose modes enabled and will perform maximum compression, even if you don't explicitly specify them.

To move a file from compressed format (compress) to gzip format, use the command:

```
znew [-t] [-v] [-9] [-K] [-P] [file.Z ...]
```

where -t tests new file before deleting original.
-v is verbose mode.
-9 uses maximum compression.
-K keeps the .Z file when it is smaller than the new .z.

-P uses pipes to bypass disk utilization, thus saving temporary disk space needs.
file.Z ... is one or more compressed files to convert.

7.1.3 Installation

The installation of the gzip binaries is the first task you must perform if you are going to use Internet software systems on your AIX platform. Because all other software packages will come in compressed format, you must have the gzip software in place before you can unwind and use or build that particular software. Because of this, you must first get and install gzip. gzip binaries have been included on the accompanying CD, or you can build the appropriate gzip binaries from the source code included on the CD.

To build the gzip binary from the source code, issue the command:

```
$ make -f Makefile.rs6k
```

By default, the gzip binary files and associated man pages will be from this procedure. If you wish to install the binaries, issue the command:

```
$ make -f Makefile.rs6k install
```

The standard directory for most GNU AIX software is /usr/local, and this conforms to the old UNIX standard of using /usr/local as the repository of local and third-party software. If you want to place the resulting binaries in a directory other than /usv/local, you can specify a command like:

```
$ make prefix=/usr/kevin -f Makefile.rs6k install
```

This will place all resulting code in /usr/kevin instead of the default /usr/local. Note that if you can't get the software from aixpdslib, you may need to run configure on the code before building with make. configure will figure out the type of machine you have and will generate the appropriate makefile. If you choose to use the binaries from the CD, once you have unwound the tar file, you can place the binaries anywhere you like. As long as the bin subdirectory is contained in your PATH variable, you will be able to find the gzip binaries.

7.1.4 Conclusion

gzip is a very powerful package for generation, control, and manipulation of compressed files. It will handle compressed and packed files as well as its own gzip format. Along with gzip and gunzip come a variety of utilities to manipulate compressed files. gzip is needed for almost all GNU distribution files since they are in gzip format and need to be un-

compressed before use. gzip is clearly one of the most powerful and widely used Internet tools. Once you have installed both gzip and the gcc system as described in the following section, you can build more current versions of gzip if they are available. gzip uses the standard configure system, which provides a relatively easy way to build free software. See Sec. 7.4 for more information on this capability.

7.2 gcc

7.2.1 Introduction

The GNU C compiler is often known as gcc and includes not only a C compiler but a C++ compiler as well. The AIX binaries for gcc are available on the accompanying CD; however, you can also get source code for any version and build it as described in Sec. 7.2.3. Once you have installed the gcc binaries, you can generate any other system including the full gcc system.

gcc is the GNU C compiler and is widely considered to be the best available, in many cases better than those available directly from vendors. Because of the wide acceptance of the gcc, it has become the de facto standard in many makefile and build procedures. This chapter certainly cannot cover all aspects of gcc; however, it will discuss the most important aspects of this very good compiler and also describe where and how to get more information. Much of the information in this section is taken directly from files delivered with the gcc product. See below for more information.

Since version 2 release of the gcc system, both the C compiler (gcc) and C++ compiler (g++) have been integrated together. This provides a compiler and preprocessor for both the object-oriented C++ environment and the C language environment. This is the only compiler system available today which has this capability. Note, however, that the run-time libraries for the g++ system known as libg++, are not delivered with this distribution and must be retrieved and installed separately.

Another thing important to note is that because of changes in gcc 2.4 and later versions, the newly created binaries are incompatible with binaries created by earlier versions of gcc. Keep this in mind if you attempt to link or use cross-compiled binaries. gcc 2.4 and later versions also supports a floating-point emulation subsystem which makes it possible to use longer floating-point types as well as to support both big and little endian environments in a cross-compiler mode. gcc 2.4 and later versions now also support Objective C, which is another object-oriented environment which provides significant value to object-oriented programmers. Finally, there are a variety of new features for

C++ which are documented in the NEWS file. Remember that to effectively use the C++ capability of gcc, you should also use the libg++ files. These files are contained on the CD included with this book.

It is not the intention of this chapter to provide programming information or tips but instead to present some of the capabilities and options that the GNU compilation system provides. While many options are discussed in this chapter, they are by no means all the options that are available. See the GNU documentation included in *Using and Porting GNU CC* (for version 2.0), which is available from a variety of sources, including the Free Software Foundation and the gcc distribution in texinfo format. See Sec. 8.3 for more information on how to get this information.

gcc is a GNU product and as such is subject to its GNU General Public License as included both in the product distribution and in App. C of this book.

7.2.2 Usage

The basic usage of the gcc compiler is:

```
gcc [options | file] ...
g++ [options | file] ...
```

where - option is one of a list of many options some of which are described below.
- file ... is one or more source input files.
gcc invokes the C compiler and uses defaults a standard C compiler would.
g++ invokes the C++ compiler and uses defaults a standard C++ compiler would.

The options for gcc and g++ are many and can be divided into major sections which are determined by functionality. The basic areas of option functionality, as defined by the GNU documentation, are:

1. Overall options
2. Language options
3. Warning options
4. Debugging options
5. Optimization options
6. Preprocessor options
7. Linker options
8. Directory options

9. Target options
10. Machine-dependent options
11. Code-generation options

Each section will be reviewed and some of the more powerful options discussed.

Overall options. The overall options section describes options which apply to the entire command line and compilation process. Some of the options are:

`-E`	Stops after preprocessing
`-S`	Creates an assembler file for each source code input file
`-c`	Creates object files but does not link to create the executable
`-o file`	Places the output in file instead of the standard a.out or *.o
`-pipe`	Uses pipes to commnicate between phases of the compilation process instead of temporary files
`-v`	Verbose mode
`-x language`	Specifies a specific language to compile where the possibilities for language are: assembler assembler-with-cpp c c-header c++ cpp-output none objective-c

Language options. These options provide different levels of support for different versions and syntax of the various supported GNU languages. Some of the possible options are:

`-ansi`	Supports ANSI standard syntax.
`-fdollars-in-identifiers`	Supports $ signs in identifiers. This is the default when the -ansi switch is not used.
`-fall-virtual`	C++ only. All member functions declared in the same class with a method-call operator method are treated as a virtual function of the given class (except constructor functions nd new/delete member operators).
`-fenum-int-equiv`	C++ only. Allows conversion of enum to int data type.
`-fno-asm`	C doesn't recognize asm, inline, and typeof as keywords.
`-fno-builtin`	Turns off support for non-ANSI built-in functions.
`-fno-strict-prototype`	This is supported for g++ only. Causes declaration statements to specifically not type function arguments.
`-fsigned-bitfields`	Specifies signed bitfields for all machine types.
`-funsigned-bitfields`	Specifies unsigned bitfields for all machine types.
`-funsigned-char`	Defines the type char to be unsigned for all machine types.

`-fsigned-char`	Defines the type char to be signed for all machine types.
`-fouriteable-strings`	Stores string constants in the writable data segment and doesn't uniquize them
`-traditional`	Includes support for Kernighan and Ritchie C syntax.
`-traditional-cpp`	Includes support for original cpp syntax.
`-trigraph`	Supports ANSI C trigraphs.

Preprocessor options. These options control the preprocessor phase of the compilation process. Some of the options available are:

`-C`	Keeps comments.
`-Dmacro`	Defines macro macro.
`-Dmacro=string`	Defines macro macro as string.
`-Umacro`	Undefines macro macro.
`-E`	Runs only the preprocessor phase.
`-H`	Displays each header file as used.
`-M`	Generates a rule which describes the relationship between all files in the compilation process. This can be used by make as a dependency listing.
`-MM`	Like -M but only describes header files included with #include.
`-P`	Does not create #line commands.
`-dM`	Outputs list of macros.
`dD`	Passes all macros to output.
`-i file`	Processes file as input discarding all output. This makes macros available to a later part of the preprocessor process.
`-include file`	Processes file before the regular input file.
`-nostdinc++`	Does not search for C++ header files in the standard directories.
`-undef`	Doesn't define any nonstandard macros.
`-idirafer dir`	Includes dir in the second include path.
`-iprefix prefix`	Specifies prefix as the prefix for subsequent "-iwithprefix" options.
`-iwithprefix dir`	Adds dir to second include path.
`-nostdinc`	Uses only include directories specified on the command line and not the standard directories.

Linker options. Linker options pertain to the link editor processing which generates executables from one or more object files. Some of the more basic are:

`-llib`	Links with the library lib. lib files are normally archive libraries (see Sec. 4.5 for more details).
`-nostdlib`	Doesn't use any standard system libraries; uses only those specified.
`-static`	Uses static libraries. Dynamic libraries are not used.
`-logjc`	Links an Objective C program.
`-nostartfiles`	Doesn't use standard system startup files.
`-shared`	Creates a shared object.
`-symbolic`	Binds references to global symbols when building a shared object.

Directory options. Directory options specify directories on which to operate. Most compilation phases have default directories and areas in which they look. These commands modify the default behavior of the compilation commands. Some of the most basic are:

`-Bdir`	Specifies a directory to search for all compilation process executables (cpp, cc1, ld, etc.).
`-I-`	Directories specified with -I- and with -Idir before -I- are searched only for #include "file" and not #include <file>.
`-Idir`	Searches for include files in dir.
`-Ldir`	Searches directory for archive library files.

Warning options. Warning options control the output of all diagnostic messages. Some of the most basic are:

`-W`	Displays extra warning messages pertaining to optimization problems, function prototyping, and possible type problems
`-Wall`	All -W switches documented below and some others as well
`-Wformat`	Checks all output statement formats and ensures output variables match output type definitions
`-Wimplicit`	Warns when a function is implicitly declared
`-Wswitch`	Warns when a switch statement is not constructed with all possible types
`-Wuninitialized`	Warns when an automatic variable is used without being initialized
`-Wunused`	Warns when a variable or function is not used
`-fsyntax-only`	Runs a syntax check on the code but doesn't compile
`-pedantic`	Issues all non-ANSI warning messages as appropriate
`-w`	Suppresses all warning messages
`-wno-import`	Inhibits warnings about import statements

There are many other possible warnings you can set when compiling and building programs with gcc and g++. See the man page gcc.1 in the main directory for more detail on these warning triggers.

Debugging options. Options which support the debugging of your codes are important and are therefore the focus of one section of this chapter. The GNU compiler provides many, some of which are described below:

`-dx`	Where x is one of the following: L—dumps after loop analysis M—dumps all macro definitions N—dumps all macro names f—dumps after flow analysis l—dumps after local register allocation m—dumps memory usage statistics at end of run y—dumps debugging information r—dumps after RTL generation x—generates RTL for a function instead of compiling it

	j—dumps after first jump optimization a—dumps all possible The dump commands provide information for stages of the compilation process. This may be used to debug gcc itself.
`-g`	Includes debugging information which can be used by gdb, dbx, or DWARF. The GNU system allows you to use -g with optimization. This is the only system available which does this, and this can be a time saver when you are moving and porting large amounts of code. The native debugging format is determined at build time for gcc. See Sec. 7.2.3 for more details.
`-gcoff`	Produces debugging informatoin in coff format.
`-gdwarf`	Produces debugging information for DWARF.
`-ggdb`	Produces debugging information for gdb.
`-gsdb`	Produces debugging information for sdb.
`-gstab`	Produces debugging information in stabs formed.
`-gxcoff`	Produces debugging information in xcoff format.
`-p`	Includes information for the prof command.
`-pg`	Includes information for the gprog command.
`-save-temps`	Saves all temporary files including .cpp and .s files.
`-print-file-name=lib`	Prints the full name of lib that would be used when linking.
`-print-libgcc-file-name`	Same as -print-file-name=libgcc.a.

Optimization options. Optimization will make your code run more quickly by rearranging statements and reorganizing the way your code works. This provide significant enhancements in the performance of your code while requiring almost no work on your part. It does have one side effect, however. By rearranging the code, you may have problems with interactive debugging because debugging information in the executable may or may not match what is being executed. GNU has addressed this and does provide the capability to debug optimized code.

With optimization options, there are many commands with begin with -f. The basic structure of these options is:

```
-foption
-fno-option
```

where the no- precedes and turns off option. For every -foption, there is a -fno-option which is exactly the opposite. Because of this, GNU documentation only describes one or the other. The option described is *not* the default. Obviously, to get the default behavior you simply add or remove the no- options as described, and you can understand the default behavior of the compiler. The basic -f options described below are *not* the defaults.

Some of the more basic options are:

`-O`	Optimizes.
`-O1`	Same as -O.

`-O2`	Highly optimizes.
`-O3`	Highest optimization including -finline functions.
`-ffloat-store`	Doesn't store floating-point variables in registers.
`-fno-default-inline`	C++ only. With this option you must member functions as in line explicitly rather than assuming it is the default.
`-finline`	Expands functions in line instead of the default of -fno-inline.
`-fsave-memo ited`	Uses heuristics to compile C++ faster.
`-fforce-mem`	Forces memory operands to be copied into registers before performing arithmetic on them.
`-fforce-addr`	Forces address constants to be copied into registers before performing arithmetic on them.
`-finline-functions`	Expands all functions into their callers.
`-ffast-math`	Optimizes math function.

There are many other -f options related to code optimization; however, they are beyond the scope of this book. They require a great understanding of the compilation process and machine architectures, which most people do not have. Examine the manual page for gcc for more information if you are interested.

Target options. You can specify a different compiler, including version and machine architecture, by generating a different target environment. This allows you to generate code for a machine architecture that is different from your current one and to use older or newer versions of the GNU compiler to generate the appropriate code. This is a very powerful feature of the GNU compiler.

Some of the basic options related to this are:

`-b machine`	Machine specifies a target machine for which to build the executable. This relates directly to the gcc installation process and its generation as a cross compiler. See Sec. 7.2.3 for more details.
`-V version`	Specifies which version of the GNU compiler to run. This is useful when you have installed multiple versions in a cross-compiler environment.

Machine-dependent options. gcc and g++ have many features that are tuned to maximize performance on a particular hardware platform. Because of this, there are many options which control the exact compilation process for any given architecture. There are sets of -m options pertinent to several different hardware architectures. Some of the more basic for the Motorola 6980x0 processors are:

`-m68000`	Compiles for a 68000 processor
`-m69020`	Compiles for a 68020 processor
`-m68881`	Compiles for a 68881 floating-point processor
`-m68030`	Compiles for a 68030 processor
`-m68040`	Computes for a 68040 processor
`-mfpa`	Compiles for a Sun floating-point processor

-mshort	Makes int 16 bits wide
-mbitfield	Uses the bitfield instruction (this is the default)
-mnobitfield	Doesn't use the bitfield instruction

For the Sun SPARC processors, they are:

-mfpu	Compiles using floating-point instructions
-mno-epilogue	Generates separate return instructions for return statements
-mv8	Generates SPARC v8 code
-mcypress	Generates code optimized for Cypress chip
msupersparc	Generates code optimized for supersparc chip

For the Motorola 88x00 processors, they are:

--mbig-pic	Compiles position-independent code
-midentify-revision	Includes ident information in the assembler output
-mno-underscores	Generates symbol names without a preceding underscore (_)
-mcheck-zero-division	Generates software traps for divide by zero exceptions
-mno-ocs-debug-info	Omits debugging information as specified by the 88open Object Compatibility Standard
-msvr4	Turns on compiler extensions which are supported in System V Release 4
-msvr3	Turns on compiler extensions which are supported in System V Release 3
m88000	Generates code for the 88000 chip

For the RS6000 machine, the options are:

-mno-fp-in-toc	Defines no floating-point constants in the table of contents; the default is -mfp-in-toc.

There are other machine-specific options for MIPS, IBM RT, Convex, and C2. See the documentation delivered with gcc for more information.

Code-generation options. These options determine interface options for code generation including function call definitions and variable and structure definitions. Some of the most basic are:

+eN	Where N is 0 or 1. +e0 declares virtual function definitions as extern and is used as an interface definition. No code is generated for this virtual function. +e1 actually generates virtual function code.
-fshort-enums	Allocates on as many bytes to an enum as are required.
-fshort-double	Makes double same size as float.
-fno-common	Places global variables in bss section rather than generating them as common blocks.

`-fvolatile`	Defines all memory references through pointers to be volatile.
`-fpic`	Generates position-independent code.
`fpcc-struct-return`	Uses the convention for struct and union values used by the default C compiler on your system.

There are many other options relating to most major compiler options areas discussed in this section. See the man page gcc.1 or documentation from GNU for more information.

The GNU C preprocessor. One of the tools distributed with gcc and g++ is the GNU C preprocessor known as cccp. This is a replacement tool for the standard ccp which comes with most C compilers. This allows you to include precompiler directives such as #include and #define to control the behavior of the code and the compiler as well as to build conditional code based on predefined macros. The basic syntax of the command is:

```
cccp [-$] [-Apredicate[(value)]] [-C] [-Dname[=def]] [-dD] [dM]
[-I dir] [-H] [-I-] [-idirafer dir] [-imacros file] [-include file]
[-iprefix prefix] [-iwithprefix dir] [-lang-c] [-lang-c++]
[-lang-objc] [-lang-objc++] [-lint] [-M] [-MG] [-MG] [-MDfile]
[-MMD file] [-nostdinc] [-nostdinc++] [-P] [-pedantic]
[-pedantic-errors] [-trigraphs] [-Uname] [-undef] [Wtrigraphs]
[-Wcomment] [-Wall] [-Wtraditional] [infile | -] [output | -]
[-trigraphs] [-Uname] [-undef] [-Wtrigraphs] [-traditional]
```

where -$ doesn't support the use of $ in identifiers.
-Apredicate(value) asserts the predicate with value (much like #assert).
-C preserves comments.
-Dname[=def] predefines name as a macro with a definition of def.
-dD generates list of #define commands and the results of C preprocessing except for predefined macros.
-dM generates list of #define commands without any results from any other commands.
-I dir defines dir as a place to search for include files.
-H displays the name of each included header file.
-I—if any -I dir options follow the -I-, all specified directories are search for both the #include "file" and #include <file> directives. Any -I dir options specified before the -I- option cause only the #include "file" directives to search in dir and not the #include <file>.
-idirafter dir adds dir to the secondary include search path.
-imacros file preprocesses file and discards all output except macros which are made available to the rest of the preprocessor process.

-include file includes file in the compilation process.
-iprefix prefix specifies prefix for following -iwith-prefix options.
-lang-c turns off C++-specific features.
-lang-c++ turns on C++ support including comment support and extra default include directories.
-lang-objc turns on the Objective C #import directive.
-lang-objc++ turns on both -lang-c++ and lang-objc.
-lint examines input code for lint command and precedes with a #pragma directive.
-M[-MG]—instead of outputting standard preprocessing, generates rules for a makefile. [-MG] says to treat missing header files as generated files and to assume they live in the same directory as the source files.
-MM[-MG] is like -M[-MG] but only outputs files included with #include.
-MD file is like -M but output is written to file.
-MMD file is like -MD but only outputs user header files, not system header files.
-nostdinc doesn't use standard include directories.
-nostdinc++ doesn't search for header files in the C++ specific directories.
-P doesn't generate any # lines as output.
-pedantic checks for ANSI C compliance and issues warnings.
-pedantic-errors issues errors rather than warning as above.
-traditional imitates K&R C, not ANSI C.
-trigraphs supports trigraph sequences.
-Uname doesn't predefine the macro name.
-undef doesn't predefine any of the standard macros.
-Wtrigraphs issues a warning if any trigraphs are encountered.
-Wcomment issues a warning when a comment is encountered.
-Wall is same as both -Wtrigraphs and -Wcomment.
-Wtradional issues a warning when encountering constructs which behave differently between ANSI C and traditional C.
infile | - is infile is the file to be processed. A hyphen (-) denotes standard input.
outfile | -—output is the output file. A hyphen (-) denotes standard output.

The C preprocessor is the first link in the compilation chain. It is typically invoked by the cc command but can be invoked separately with the ccp or cccp command.

cccp provides the function of including header files, expanding macros, and providing for conditional compilation. It is often used to gener-

ate machine-dependent sections in source programs where the end result is that only the appropriate lines of code are passed to the C compiler itself.

There is a man pages cccp.1 included in the gcc distribution which has more detailed information about the options and cccp itself. See this for more documentation.

g++. g++ is the C++ compiler which comes with gcc. It used to be a script which passed the correct options to gcc to invoke the C++ portion of the gcc compiler. However, it is now a full-blown C++ compiler and a C program. Its options are very similar to those supported by gcc with a few exceptions. The man page g++.1 is a good listing of the issues and brief commands related to the g++ compiler. See this for more details on g++.

The other issue related to g++ is your need for libg++. This is a group of class libraries which complement gcc and g++. There are other directories including libiberty which contain a variety of commonly used GNU files. See Sec. 7.3 for more information.

Debugging issues and gcc/g++. GNU distributes a debugger known as gdb. Section 7.10 outlines how gdb works and what kind of information it expects. This is relevant to the gcc section because the gcc system is responsible for generation of the appropriate information so that gdb can function correctly.

The standard debugging system on UNIX is dbx, which comes native with most flavors of UNIX. You can use dbx on code compiled with gcc; however, gdb has certain features which work better and more effectively with gcc-generated code.

One issue directly related to this is the special debugging output format known as DWARF. DWARF is the standard in System V Release 4 to define standard debugging formats. gcc supports DWARF V1 except in some instances which may cause incompatibilities with SVR4 SDB. If functionality is replaced by DWARF V2, it is most likely included in gcc, replacing the DWARF VI implementation. These are documented in the file README.DWARF in the gcc distribution. There is work in progress to finish the DWARF V2 specification, and when this is finished, you can bet that gcc will work toward support of this standard. In fact many of the features of the early work on DWARF V2 are already incorporated in gcc.

There is not compatibility for DWARF with g++, and there may be none in the near future since there is some considerable debate about the need for and relevance of DWARF to C++. Watch the distribution information with new versions of g++ for more details.

Using gcc/g++ as a cross-compiler. One of the most powerful features of the gcc/g++ system is its ability to act as a cross-compiler. This means that gcc on one machine can generate code for a different machine architecture. For example, if you are running on a Sun, you can generate an executable that runs on an IBM RS/6000. This is extremely powerful and provides multiplatform support from a single machine architecture for not only source code but object and executable code as well. A simple example of this is given in the Examples portion of the next section and some discussion is made in the next section about how this is accomplished.

For each machine architecture you wish to support, you must install a version with the configure command and note the target or host architecture. This means that you will have a completely separate version of gcc/g++ for each architecture you wish to compile code for. For example, if you are running on an HP machine and plan to distribute executable code for a SUN and IBM RS/6000 as well, you will need to install and configure three separate versions of gcc on the HP machine if you wish to perform cross-compilations from one HP.

Now that you are excited about the possiblity of performing cross-compilation, let's have a brief discussion of what you need to accomplish this. In addition to the gcc system, you need to provide both a cross-assembler and a cross-linker. These tools are available from a variety of vendors and can be used with gcc to create cross-compiles. You also need the appropriate header files for a particular machine so that linking can occur correctly. None of these is a "show stopper" but will require some additional work with respect to getting the appropriate tools to provide the cross-compilation capabilities. See the section on cross-compilation in the INSTALL document for more guidance.

This is a very powerful feature of the gcc system and one which you should take advantage of if you are running in a heterogeneous environment of UNIX plaforms. There are several ways you can get the proper tools assembled to create this capability. Don't let the complexity stop you from pursuing this capability if it is one you really need. See the next section for more information.

7.2.3 Installation

This section will discuss installing gcc from the source code. It is included on the CD in executable format as well. See the notes on the CD for relevant issues for installing the executables.

The installation of gcc will create various directories including /usr/local/lib/gcc and /usr/local/bin if they don't already exist (unless of course you have redefined prefix with some other directory). You can redirect the result of the builds with specific commands; however, the

default is to write to the /usr/local area. Check your product specifications and standards before installing gcc.

Once you have unwound the tar file, you will notice that there are several files related to the installation of gcc. Some of the more important are:

INSTALL

README

README.*, where * stands for a variety of supported platforms including ALTOS, APOLLO, DWARF, MIPS, NS32K, RS6000, TRAD, and X11

ChangeLog.*

NEWS

The INSTALL file contains notes relevant to the building of gcc, while the README files contains specific information on particular platform issues. Inside the INSTALL file, there is a discussion of a variety of machine-specific issues. The ChangeLog files contain information on release information and functionality. The NEWS file contains "noteworthy changes" in the more recent versions of gcc and g++. Examine these files before proceeding.

The general build recommendation for gcc is the same as for most other GNU commands. If you have built any other versions of gcc, you first need to ensure that you have removed all executables and machine-dependent scripts with the command:

```
$ make clean
```

Now you can run the configure tool, which will build the appropriate makefiles and install scripts for your platform. There are a variety of platforms for which you can configure gcc. See Sec. 7.4 for more information on exact options and machine types for the configure command.

There is a bug in the RS/6000 assembler in versions earlier than AIX 3.2.4. If you are going to use the -g option on the cc, you have PTF U416277 installed on your machine or the build will not proceed correctly.

If you are running AIX 3.2.4 or later, the fix is already in place. To check if this fix is in place, type:

```
$ as -u < /dev/null
```

If you get an error about "-n* unknown flag" you need to apply this PTF; if not, you can proceed without this.

If you are not interested in debug information, remove the -g option from the CFLAGS macro and the -g1 in the LIBGCC2_CFLAGS macro in Makefile.in before you run the configure command. You also need to remove the two occurrences of -g0 in the crtbegin.o (line 739) and crtend.o (line 743) targets. If you do not do this, you will get an assembler error that only the PTF mentioned above will fix.

There is also a problem building gcc with XLC 1.3.0.0 which ships standard with AIX 3.2.5. You need to get XLC 1.3.0.1 or later to build gcc from scratch. To get this, get PTF U432238 and install it.

On an RS/6000 machine, you might use a command like:

```
$ limit mem u
$ limit data u
$ configure --prefix=/usr/local-build=rs6000
```

The first two commands ensure that you won't run out of memory or data space. If you don't use the --prefix option, the configure will set up the default directories for the binaries and library files that result from the build to be in /usr/local/bin and /usr/local/lib. If you don't want to do this, use the --prefix and tell configure where you want the make to create the files. If you are uncertain as to your platform type, use the default generated by configure. If the type generated by the configure system is incorrect, match the one which best fits your system, and you may have to modify the Makefile created before you can successfully build your product. This should not be the case for gcc.

Once you have created the proper build files, you need to examine the dates on the files c-parse.y, c-parse.c, cexp.y, and cexp.c. If the dates on the .c files are later than the dates on the .y files, you can proceed. If this is not the case, you need to either invoke yacc to rebuild the .c files or use the bison system, which is the GNU equivalent to yacc. See Sec. 7.7 for more details on GNU bison.

If you are using any other GNU tools such as the assembler (gas) or the leader (gld), you will need to install them before proceeding. You can specify these tools instead of the native ones with the --with-gnu-as and --with-gnu-ld options on the configure command listed above.

Build the compiler by first moving to the main directory and invoking the make command. For example:

```
$ make LANGUAGES=c CC=/usr/local/bin/gcc
```

This builds only the C component of the compiler. This is the recommended technique if you are using anything other than the previous version of GNU C compilers to build gcc. Many vendor-supplied compilers will not build gcc and g++, and therefore you may want to first build only the C compiler to minimize the number of possible build problems.

If you are using a previous version of gcc, make sure you fully quality the gcc pathname to the makefile to ensure you don't get a partially built version of gcc from the current directory.

Once you have built gcc with the above commands, you have built what is called the stage 1 group of the compiler system. This builds a minimal implementation of the gcc system which you can then use to build the entire gcc and g++ system. To take the results of the phase 1 compilation and move them to proper subdirectory, issue the command:

```
$ make stage1
```

This will create a stage1 subdirectory which contains all the necessary files to continue the gcc installation. At this stage, it is appropriate to create and move any other GNU executables that you may need to build the rest of the products into the stage1 subdirectory. An example tool is gas (GNU Assembler and GNU linker). If you don't want to do this, you can modify the PATH variables to ensure that the GNU tools (as and ld) precede the standard vendors' versions of these products. This ensures that gcc finds these tools as it continues to build.

Next you should recompile the gcc compiler itself with a full implementation of the stage1 object files and executables. To do this, type:

```
$ make CC="stage1/xgcc -Bstage1/" CFLAGS="-g -O"
```

Note that the default is to build everything; however, you can specify one or more of C, c++, or objective-c to build those individual products. This means that the LANGUAGES variable above is unnecessary since this is the default. However, you can specify one or more of the LANGUAGES listed above to build only those languages.

This generates what is known as the stage 2 version of the products. With this you can generate what GNU calls the stage 3 products by issuing the the same make command as above but substituting stage2 for stage1 in the above string. Remember to include gas and ld as in stage 1. For example:

```
$ make stage2
$ make CC="stage2/xgcc -Bstage2/" CFLAGS="-g -O"
```

This will create a stage3 subdirectory which contains identical information to the stage2 subdirectory. With this you can compare the stage2 and stage3 subdirectories with the command:

```
$ make compare
```

If you are using Objective-C, now is the time to build the Objective-C libraries with the command:

```
$ make objc-runtime CC="stage2/xgcc -Bstage2/" CFLAGS="-g -O"
```

This will generate the Objective-C libraries in the appropriate directories.

To install the created files, issue the command:

```
$ make install CC="stage2/xgcc -Bstage2/" CFLAGS="-g -O"
```

This installation procedure copies the gcc executable to the /usr/local/bin directory and cc1, cpp, and libgcc.a to /usr/local/lib/gcc-lib/TARGET/VERSION where TARGET is that specified to configure and VERSION is the version of gcc you are building unless you specified a --prefix on the configure or make command above. If you used the --prefix directory, the make install will install the binaries in the prefix directory in a bin subdirectory (as in the case described above), the libraries in the prefix directory in a lib subdirectory, and so forth. This is a powerful capability which allows the products to be grouped closely with the source code and fits with the overall software structure and philosophy discussed in this book. It will be necessary to include the correct directory in the PATH variable for the gcc executable to ensure that all other scripts and product builds know of the existence of gcc.

You can specify a different directory for these files with the libdir and prefix switches on the make statement as documented above. Experiment with these variables if you need to modify the location of these files to conform to your product standards.

If you have problems with the installation related to include file syntax, try the command:

```
$ make install-fixincludes
```

Install the Objective C portion of the compiler with the command:

```
$ make install-libobjc CC="stage2/xgcc -Bstage2/" CFLAGS="-g -O"
```

If you want to use the g++ capability of this distribution, you need to install the libg++ run-time libraries.

If you have problems, you can always type:

```
$ make clean
```

If you want to use gcc as a cross-compiler system, you must repeat the above process using a different target architecture in a different

directory structure to create a gcc version which will generate code for this different architecture. Note that there are several problems with this, not the least of which is that you need to provide cross-linkers and cross-assemblers. gcc generates assembler code which must be assembled and linked for the appropriate platform. You must provide those cross-assembler and linker files. They are available from a variety of vendors, including those who develop embedded control systems and on-line systems. If you can get this technology, gcc will provide a transparent executable portability and cross-compilation system. See the INSTALL document for more information on cross-compilation.

Once you have installed the version for the particular architecture, you can use machine-specific options (-m...) for that platform as you would if that were the native implementation for your current hardware platform.

When you have successful installed the product, you will want to remove any excess directories because of their large size. Issue the command:

```
$ rm -r stage1
```

stage2 will contain the correct executables for gcc and ensure that you have the minimum numbers of files to reconstruct gcc should you need to.

There are known problems with using gcc. See the PROBLEMS file for more information.

Examples. Some simple examples of invoking gcc follow. To invoke the compiler with ANSI C language support, type:

```
$ gcc -ansi -o kevin.exe kevin.c
```

which processes kevin.c and generates an executable named kevin.exe.

To invoke the compiler and build object files without linking the final executable, type:

```
$ gcc -c file1.c file2.c
```

The result of the above command is file1.o and file2.o.

If you want to see the preprocessed code as it would be passed to the compiler, use the command:

```
$ gcc -E kevin.c > kevin.output
```

and examine kevin.output, which will contain all preprocessed information and the original source code.

To use directories that are different from the standard for include files, use a command like:

```
$ gcc -I /usr/local/kevin/lib kevin.c
```

which will cause gcc to search in /usr/local/kevin/lib for include files instead of in the standard directories.

To generate files used for debugging, type:

```
$ gcc -g kevin.c
```

which will create a file a.out, which includes debugging information which can be use by gdb, dbx, or other debuggers.

Finally, if you want to highly optimize your code after you are confident it is working correctly, type:

```
$ gcc -O2 kevin.c
```

This will generate an a.out which runs faster than the nonoptimized version.

To invoke gcc as a cross-compiler, use the -b switch as follows:

```
$ gcc -b sparc-sun-solaris kevin.c
```

This will create an a.out file which can run on a Sun machine even though you may have compiled it on a non-Sun platform. See the section "Using gcc/gtt as a Cross-Compiler," above, for more details.

Service. There is a file named SERVICE which provides a listing of people and organizations which provide support and assistance for gcc. See this file if you are interested in additional support for your gcc system above and beyond what you get from GNU and the Internet community.

7.2.4 Conclusion

Much of this information was gleaned from the manual page that comes with gcc. See this manual page and the associated texinfo documentation for more details.

gcc is one of the best C and C++ compilers available today, and it runs on virtually every platform. With the Version 2 distribution you get both a C compiler and a C++ compiler. This provides incredible functionality while maintaining relative simplicity in terms of use and installation. It should be noted that g++ is a real C++ compiler and not simply a front-end tool which converts C++ to C and then compiles it.

This provides a much better and more powerful capability and subsequent code than some of the other C++ front ends available on the market today.

There are many options, most of which you will never use. However, it is nice to know that this kind of functionality is available if you need it. There are more options available than are documented in this chapter. See the delivered documentation for more information. gcc/g++ is one power tool which you should definately investigate.

7.3 libg++

7.3.1 Introduction

libg++ is a collection of libraries which are commonly used by g++ and gcc for compilation and construction of programs. Several other libraries come with this distribution, including the liberty library of free software. This consists of a collection of commonly used GNU routines and class libraries. There is a significant amount of documentation included with this distribution in the form of texinfo files. See Sec. 8.3 for more information on texinfo and how to access this information.

libg++ is a GNU product and as such is subject to its GNU Library General Public License as included in the product distribution and in App. C of this book.

7.3.2 Usage

The usage of libg++ is really from tools like g++. There are no interactive commands which you can execute to produce anything of value. The value of libg++ is the class libraries included with the distribution. There are many class libraries and examples of C++ source code contained in this distribution. Some of the classes in libg++ 2.6 are:

IOStream	List
Stream	LinkList
Obstack	Vector
AllocRing	Plex
String	Stack
Integer	Queue
Rational	Deque
Complex	PQ
Fix	Set
Bit	Bag

Random	Map
Data	GetOpt
Curses	Projects

These classes provide numerous capabilities to work with a variety of different objects with the g++ compiler. There is more documentation in the .info files in the libg++ subdirectory. See this and Sec. 8.3 for more detail.

7.3.3 Installation

The first file to examine in the distribution is the ./libg++-2.6.2/README file. This contains various pieces of important advice that you should be aware of before attempting to build the product. Some of the more important points are:

Use gcc to compile libg++, and use a version of gcc that is at least as high as the version of libg++, in this case gcc 2.6.2 or greater (preferably 2.6.3).

If you haven't installed the gcc compiler in the expected /usr/local area, you must create a symbolic link in the main directory (one level above the libg++ subdirectory) which points directly to the gcc executable so that the make can find gcc.

Don't use GNU sed 1.12 or you will have build problems.

The installation of libg++ is very straightforward and uses configure like most other GNU programs do. At the top directory of the libg++ distribution (probably libg++-2.6.2), type the following to generate the proper Makefiles for your machine:

```
$ ./configure rs6000
```

The other issue is to make sure configure can find the gcc compiler since this is important to a successful build. If you have used the make install for gcc, the configure will know where to look; however, if you have placed gcc in a directory other than /usr/local/bin, you will need to create a symbolic link in the libg++-2.6 subdirectory (one level above the libg++ directory). An example of an ln to create a symbolic link is:

```
$ ln -s /usr/local/gcc/gcc-2.6.2/bin/gcc gcc
```

This creates a symbolic link in the current directory (in this case /usr/gnu/libg++/libg++2.6.2/bin) named gcc which point to the gcc compiler created in the previous section of this chapter. It appears, on the

surface, that this may not always work and that you may have to place a similar symbolic link in the /usr/local/bin for gcc to ensure that configure for libg++ finds gcc to include in the Makefile. Without this, libg++ may not use gcc and may not compile correctly. Watch out for this.

Once you have made the gcc compiler available to configure, you can configure your machine. For example, on an RS/6000 running AIX, you would type:

```
$ limit mem u
$ limit data u
$ ./configure --prefix=/usr/local rs6000
```

The first two commands ensure that you don't run out of either memory or data space. This will create a Makefile which you can then execute. If you are using an RS/6000, you will run into the same compilation problem you had with gcc related to the use of the -g debug option. You must remove all references to the -g options in the resultant Makefile or Makefile.in. If you don't, you will get a fatal compiler error. You can eliminate this error by calling IBM and requesting PTF U416277. As before, if you are running AIX 3.2.4 or later, this is fixed for you. Once you have fixed this issue, you can proceed with the commands:

```
$ make CXX="gcc -Wa, -u -O" cc="gcc -O"
```

This will compile all included libraries in libg++. Note that you must have compiled the C++ sections of gcc to build libg++. If you haven't compiled all object-oriented parts of gcc, you will get error messages regarding a lack of availability of cc1plus. If this is the case, see the section on gcc above to find out how to build full-blown gcc.

Optionally, you can now type:

```
$ rm /usr/local/lib/g++-include/*.h
$ make install
```

if you want to place the resultant files in the /usr/local/lib areas or wherever you defined your prefix area.

Finally, you can test your results with the command:

```
$ make check
```

This will generate and execute some test to ensure that your build process was successful. If you have errors, you may need to rebuild from scratch. If you continue to get errors, see the documentation for potential problems.

7.3.4 Conclusion

libg++ contains a variety of class libraries from curses to iostreams to data types. By using these class libraries as well as the other library files provided with libg++, you will get a good look at some excellent class libraries and applications you can build with them.

7.4 configure

7.4.1 Introduction

GNU has begun distributing most of their software with a tool called configure. configure examines your machine and structures all appropriate build tools such as makefiles and install scripts for the proper machine and software type. One of the biggest problems with Internet software has been the installation. Because of the differences between UNIX implementations, often building software for a particular "flavor" of UNIX has been difficult if not impossible. Only with great modifications to either the source code or the makefiles were you able to build the software on a given platform. GNU has attempted to fix much of this problem with their configure tool.

configure prepares a source code system to be built. This tool will save hours of activity and frustration by providing a consistent and robust software-building interface.

7.4.2 Usage

The basic syntax for the configure command is:

```
configure HOST [--target=TARGET] [--srcdir=DIR] [--rm] [--site=SITE]
[--prefix=DIR] [--exec_prefix=DIR] [--program_prefix=DIR]
[--tmpdir=DIR] [--with-PACKAGE[=YES | NO] ] [--norecursion] [--nfp]
[-s] [-vV] [--version] [--help]
```

where HOST is the name of the host machine on which to target.

--target=TARGET builds sources for TARGET and not for the default of the current machine.

--srcdir=DIR looks for source code in DIR directory.

--rm removes the current configuration; doesn't create one.

--site=SITE uses specific makefiles for SITE.

--prefix=DIR defines location of install files (default is /usr/local).

--exec_prefix=DIR sets the root directory for host-dependent files.

--program_prefix=DIR configures installation files to install programs in DIR.

--tmpdir=DIR defines directory for temporary file creation.

--with-PACKAGE[=YES/NO] sets a flag for the build to recognize the existence of PACKAGE (the default is yes).
--norecursion configures only the current directory.
--nfp specifies the lack of floating-point units.
-s suppresses status messages.
-v is the verbose option.
-V displays configure version number.
--version is same as -V.
-help displays usage summary.

Most INSTALL files describe the correct usage of configure for each GNU package distributed.

TARGET descriptions. The basic format of the TARGET descriptions for the different supported platforms is CPU-COMPANY-SYSTEM, where each of the three fields can be represented by one of a set of values. The values for the CPU field are:

```
a29k, alpha, arm, cN, clipper, elxsi, h8300, hppa1.0, hppa1.1, i370,
i386, i860, i960, m68000, m88k, mips, ns32k, pyramid, romp, rs6000,
sh, sparc, sparclite, vax, we23k
```

For COMPANY, they are:

```
alliant, altos, apollo, att, bull, cbm, convergent, convex, crds,
dec, dg, dolphin, elxsi, encore, harris, hitachi, hp, ibm,
intergraph, isi, mips, motorola, ncr, next, ns, omron, plexus,
sequent, sgi, sony, sun, tti, unicom
```

For SYSTEM, they are:

```
aix, acis, aos, bsd, clix, ctix, dgux, dynix, genix, hpux, isc,
linux, luna, lynxos, mach, minix, newsos, osf, osfrose, riscos, sco,
solaris, sunos, sysv, ultrix, unos, vms
```

The COMPANY field can be ignored if you can uniquely identify the target system with the CPU and SYSTEM fields. For example, rs6000-aix is unique and ibm is not required.

You can add version numbers to the end of SYSTEM to more specifically define a particular target system. For example, you can use rs6000-ibm-aix31. There is no guarantee that specifying a particular version of an operating system will generate a different result than nonspecification, but it can't hurt.

There are also aliases for the CPU-COMPANY combination. Some of the more popular are:

```
3300, 3b1, 3bN, 7300, altos3068, altos, apollo68, att-7300, balance,
```

```
convex-cN, crds, decstation-3100, decstation, delta, encore, fx2800,
gmicro, hp7NN, hp8NN, hp9k2NN, hp9k3NN, hp9k7NN, hp9k8NN, ifis4d,
iris, isi68, m3230, magnum, merlin, miniframe, mmax, news-3600,
news800, news, next, pbd, pc532, pmax, ps2, risc-news, rtpc, sun2,
sun386i, sun386, sun3, sun4, symmetry, tower-32, tower
```

There are a variety of bugs and behaviors for each of the possible configurations which are documented in the INSTALL file which comes with the documentation. Some examples are:

- elxsi-elxsi-bsd	There are current known problems building gcc; contact mrs@cygnus.com for more details.
- m88k-svr3	There are problems with the typically shipped Green Hills C compiler which suggest you should use a previous version of GNU C to generate V2.
- m88k-svr4 - ns32k-sequent	You need to create a file called string.h containing #include <string.h>.
- rs6000-*-aix	There is a problem with the IBM assembler. See the file README.RS6000 for more information on this problem.

You can check for the existence of a particular machine configuration type with the command:

```
$ config.sub name
```

where name is a machine name such as sun, ibm, hp, etc. This is a very useful to test before beginning the build process on your system.

The other important feature to focus on is the -exec-prefix option. This provides you with the ability to place the binary files in a different directory structure from the non-architecture-dependent files that can either be in /usr/local or in a directory specified by -prefix option. This is useful if you are exporting a filesystems which contains these programs in a heterogeneous environment. For example, exporting /usr/local to Sun, HP, and IBM machines can be easily supported with the exec-dir option.

Finally, there is a common makefile label throughout most free software packages which use configure. After you have used configure to build a system, you may want to install it. Most packages will support this methodology, and it is well described in the coming sections and chapters. But to summarize, the recommended methodology in most cases is:

```
$ ./configure
$ make clean
$ make target-machine
$ make prefix=/usr/local install
```

The configure builds the appropriate makefile. Next, the make clean removes any possible files which have been built on other machines.

This ensures that you get a clean compile and everything is rebuilt from scratch. Next, the make with the target-machine option will create any binaries and library files which are necessary to run the product. Usually, these files will be created in the local subdirectory hierarchy (e.g., ./ghostscript/ghostscript-2.6). This allows you to test the executable before installing it in a directory for public use. The final command installs the product in a specified directory (the prefix variable) or in the default /usr/local subdirectory.

Some makefiles are different, and you should see the appropriate makefile before you build your product to ensure that you are placing files where you want them to go.

7.4.3 Installation

Since configure comes with most newer GNU systems, there is no need to install it as a separate product. Therefore, this section is really unnecessary, and in fact configure is not distributed as a separate product.

7.4.4 Example

There are many examples in this book which use configure, and a fairly standard methodology for using configure has been established. Because of its frequency, a simple example using configure is displayed here for demonstration purposes. You will learn more as you build and configure other tools from GNU in other sections of this book.

To configure a product for an AIX machine and include the executables in a subdirectory named bin, use a command like:

```
$ configure --target=rs6000 --prefix=/usr/local
$ make clean
$ make
$ make install
```

This example will generate the appropriate makefile for a RS/6000 running AIX. Next the $make clean command removes any executables and libraries which may cause the make to function improperly. Finally, the $make will build the product and potentially move files into the prefix directory structure. Finally, the $make install command causes the product to be copied into the /usr/local subdirectory and causes the manual pages as well as the libraries and executables to be generated and stored in the appropriate subdirectory. Again, this is a slightly different methodology than documented earlier; however, the results should be the same. Note that the make command has the prefix directory defined in it due to its definition on the configure command. This is why the prefix option is not necessary on the make

command line. Remember that most of the time the make command will not copy (install) the resultant files unless you explicitly use the install option; however, this is not always the case, so beware.

Note that, as this example shows, you can use configure in combination with make to generate the appropriate executables and library files. You can either issue options like prefix with the configure command or the make command. Both are equally correct and will accomplish the same thing.

The final thing to note about the install with configure is that it seems to have trouble with directories that aren't there. This means that if you need any subdirectories, particularly those related to your prefix path, you should create them before you issue the make install command or you may get some nasty error message that install failed. If you see these, look carefully at the directory in which install was trying to operating and make sure it exists.

The configure command will become clearer as you use it to build products throughout this chapter and others in this book. See the following sections for more examples.

7.4.5 Conclusion

configure is a very poweful tools distributed by GNU to ease the porting and building of tools on multiple platforms. By removing and hiding inconsistencies between UNIX platforms and compilers, configure makes porting and building software on different architectures simple.

Some older versions of GNU tools don't have the configure programs, and you will have to check makefiles and source code for compatibilities and inconsistencies. While configure will not fix all problems related to multiplatform support, it removes the majority of problems and makes porting relatively straightforward.

It is a good idea to check for the existence of newer versions of software if the program does not build correctly the first time on a machine. Most Internet products have been ported to a large variety of UNIX platforms, and if you experience trouble with the build, check for a newer version of the product before wasting too much time messing with the build procedures of the current product.

7.5 make

7.5.1 Introduction

gmake (now called make) is, as you would expect, GNU's version of make. Make is a tool used for software development and maintenance. It allows you to build only the parts of a program which have changed. By building a makefile which contains information on which files are a part of the final executable and describing the structure of those files

and how they fit together, make can prevent a considerable amount of errors and inconsistencies later in the software development process. When you change one piece of an application, make rebuilds only those pieces that changed and leaves the rest untouched. This save a considerable amount of time over rebuilding the entire executable after each change. make is extremely powerful for software developers and GNU fully supports make functionality. See Sec. 5.4 for more detail on make.

There are some fixes and new capabilities with make 3.70 over earlier releases. See the NEWS file for more details.

gmake (or make) is a GNU product and as such is subject to its GNU General Public License as included both in the product distribution and in App. C of this book.

7.5.2 Installation

The installation of the make system is straighforward.

As is often the case with Internet software products, you should first print out the README file in the main directory. And as with most GNU products, you must first configure the products for your particular environment. In the case of AIX, the configure system is intelligent enough to build the system correctly. To build the system, use the commands:

```
$ ./configure cc="gcc -traditional"
$ make -f makefile.rs6k
```

Note that you can build make with either native XLC or you can use gcc. Of course on AIX 4.1 you don't have a native C compiler, so you will have to use gcc unless you have purchased the separate C compiler for AIX 4.1.

Next, to install the product type:

```
$ make install prefix=/usr/local
```

This is all there is to building and installing gmake. You can now proceed to other sections to learn more about gmake.

There are a few other subtle things you can do with the configure if you want. See the INSTALL file for more information if you are interested.

7.5.3 Usage

The syntax for the gmake command is:

```
gmake [-f makefile] [option] ... target ...
```

where -f makefile defines a makefile other than the default makefile or Makefile files.
option consists of one or more of the following options:
-C dir—changes directory to dir before searching for makefile.
-d—debugging mode. Gives more information about files and processing.
-f file—uses file as makefile input.
-i—ignores all errors.
-I dir—similar to compiles; use dir to search for included makefiles. Can be used more than once on a single make command to search multiple directories.
-j jobs—specifies the number of jobs you want to run simultaneously.
-k—continues after an error has occurred. Files unrelated to the failed target will still be made.
-l load—no new jobs will be run if there are other jobs running and the average load on the systems is load (in percent).
-n—prints out commands and results but doesn't execute them. Useful for debugging.
-o file—doesn't remake file even if the .o file is older than its associated source file.
-p—prints database that results from reading makefiles. This generates a schema for the building and stucture of your program. To print the default database, use the -f/dev/null with the -p switch.
-q—prints zero if targets are up to date and nonzero if they are not. Does not execute any commands.
-r—doesn't use implicit rules including suffixes.
-s—silent mode. Doesn't print out any commands as they are executed.
-t—touches files to fool make and prevent recompilation.
-v—prints the current version of make.
-w—prints working directory.
-W file—tells make that file has been modified even if it hasn't.

One of the big differences between gmake and most other make facilities is its support of the concept of relative path builds. The variable VPATH is often discussed in many of the configure scripts used to build many of the products discussed in this book. The configure utility uses this capability to generate binaries in directories other than where the source lives. gmake also has much richer support for substitution reference functionality.

The other very nice feature is the jobs support (-j). gmake will allow you to create and execute multiple jobs or commands at the same time.

It will automatically generate a number of job streams to execute the build more quickly by splitting the compilation job into multiple streams which are independent of each other. You can limit the load on the machine with the -l option. Explore this feature if you have the opportunity to create multiple job streams from a single makefile.

A simple example of creating a multistream job is the following command:

```
$ make -j 3 all
```

This will attempt to create up to three jobs to compile all executables and link them for a final executable. If you have a large number of file compilations which must occur before a link, this is a good strategy to speed up the build process since you can be busy compiling while another compile job is waiting for I/O. This can significantly speed up a build if used properly. Note that you don't need to do anything special to your makefile in order to take advantage of this capability. gmake will do as much as it can transparently. This is a simple example of the power of gmake.

7.5.4 Conclusion

GNU make is a fully compatible version of make that runs on a variety of platforms. It is generally seen as at least as good as the default version of make on your machines and often is better. There are often subtle differences in make functionality and behavior between machines and operating systems that are not discovered until after the project has begun. With GNU make, you can avoid many of the pitfalls and problems associated with different flavors of UNIX by using GNU make on all platforms associated with your software development.

Finally, by using advanced functions of gmake such as multiple job stream support and relative path support (VPATH), you can make your build processes quicker and better.

7.6 flex

7.6.1 Introduction

The latest release of flex is 2.4.6, which has a variety of fixes that help the compile process and greatly simplify the configuration of the flex product. The NEWS file contains the details of the changes between various releases of flex 2.4.6. See this file for more details.

flex is a replacement for lex. While it is significantly faster than standard lex, it does not maintain complete backward compatibility with lex, and therefore, you must be careful.

flex stands for fast lex and is a fast lexical analyzer. It allows you to scan and operate on input and look for patterns on which to operate. A typical use of lex and flex is to generate parsers of text which allow you to write relatively simple and straightforward routines which will analyze input information and structure the corresponding output according to rules defined in the flex input file.

The resulting output of a flex input file is a C source code routine named lex.yy.c which defines a routine yylex(). This routine can be compiled with a standard C compiler and the -lfl option in the C compilation statement. When the resulting code is executed, it parses the input and applies the rules from the flex input file to process the input and generate the appropriate output.

In-depth lex and flex syntax is not the topic of this chapter; however, the basic syntax and operations of flex will be presented to familiarize you with its basic operation and functionality. See Secs. 5.5 and 5.6 for more information about them.

flex is a BSD-related product and as such is subject to the copyright restrictions of The Regents of the University of California. A copy of the license is included both in the product distribution and in App. C of this book.

7.6.2 Usage

The basic syntax for flex is:

```
flex [-FILT8bcdfinpstv -C[Fefm] -Sskeleton] [file...]
```

where -F uses the fast scanner table.
-I generates an interactive scanner.
-L tells flex not to generate #line directives in lex.yy.c.
-T is trace mode.
-8 generates an 8-bit scanner.
-b generates backtracking information to lex.backtrack.
-c is the null option (useful for backward compliance only).
-d is debug mode.
-f is full table or fast scanner mode.
-i generates a case-insensitive scanner.
-n is the null option (useful for backward compliance only).
-p is the performance report to standard error.
-s supresses unmatched scanner input to standard output.
-t displays output to standard output instead of to the default lex.yy.c.
-v generates summary statistics to standard error.
-C[Fefm] is the table compression commands where:
F—uses alternate fast scanner representation.

e—generates equivalence classes.
f—generates full scanner tables.
m—generates meta-equivalence classes.
no option generates compressed tables but no equivalence or meta-equivalence classes.
-Sskeleton overrides default skeleton scanners file.
file... is one or more files on which to operate.

flex generates programs it calls scanners. These scanners (typically named lex.yy.c) contain code which recognizes lexical patterns. The input file to flex is called the rules file. It contains pairs of directives which consist of regular expressions called patterns and corresponding C code sequences called actions. When the resulting scanner is compiled and executed, it scans input for matching regular expressions and executes the corresponding C code.

flex is typically used for input parsing and manipulation. See Sec. 6.6.3 for some examples.

While regular expressions have been discussed in other sections of this book, flex uses an extended set of regular expressions for its rules file, and it is useful to summmarize their basic syntax here:

`c`	Matches any character 'x'.
`.`	Matches any character except newline.
`[abc]`	Matches a, b, or c in that order.
`[abd-f]`	Matches a, b, or any character from d through f.
`[^A-Z\t]`	Negated character class. This means any character but A through Z and a tab.
`regexp*`	Zero or more regular expressions.
`regexp+`	One or more regular expressions.
`regexp?`	Zero or one regular expression.
`regexp{2,5}`	Two to five regular expressions.
`regexp{2,}`	Two or more regular expressions.
`"string"`	Literal match of string.
`\num`	Where num is an octal number.
`\xnum`	Where num is a hex number.
`regexp1regexp2`	Regular expression 1 followed immediately by regular expression 2.
`^regexp`	Regular expression at the beginning of a line.
`regexp$`	Regular expression at the end of a line.
`<<EOF>>`	End of file.
`<s1,s2>regexp`	Regular expression when start condition s1 or s2 is realized.

The precedence of these operators is exactly as in other regular expression operators, and they are listed in order from highest to lowest precedence above.

The basic format of the rules file is:

```
definitions
%%
rules
%%
code
```

where definitions consist of the variables and function definitions that you would normally place in the global declaration section of your code. You can also include name definitions, which allow you to define symbolic names for regular expressions. For example, DIGITS [0-9] and LOWCHARS [a-z] are common name definitions. These can later be referenced with:

```
{name}
```

syntax. For example:

```
{DIGITS}
```

would reference the regular expression (0–9). There are almost an infinite number of things you can do with this section. It is really application dependent.

The rules section consists of pairs of patterns and actions. Patterns must begin in the first column, and the associated action must begin on the same line as the pattern. There are a variety of rules pertaining to pattern matching, including negated character classes matching newline characters unless explicitly stated and the unique occurrence of beginning of line and end of line operators. See the flexdoc.1 file for more details.

The action part of the rules section consists of pure C code. You can also include several flex directives such as:

`ECHO`	Displays token on the screen
`BEGIN`	Places scanner in a start condition
`yymore()`	Appends matched token to current yytext value
`yyless(n)`	Returns all but first n characters of the current matched token to the input stream to be rescanned
`yyterminate()`	Acts as a RETURN statement
`unput(c)`	Places character c back to the input stream
`input()`	Reads the next character from standard input

flex uses a token-based mechanism which defines token by strings separated by white space. This is a common methodology for a parser such as flex. All commands in the resulting scanner work on the current token. The token is made available to the program as a pointer yytext. The length of the token is contained in yyleng. This mechanism

can be effected with the -I option, which generates what is known as an interactive scanner. This means that instead of the default action to read ahead one character before matching a pattern, the scanner will only do this when absolutely necessary. This saves processing time and makes interactive usage more efficient. The general rule stated with respect to this is to use -I if the input is going to be interactive; otherwise, don't use -I.

The default rule is that of simply displaying input text on the output stream. The flex input file looks like:

```
%%
```

Note that if you compile and execute this, it will simply echo all that you type in.

Finally, the code section consist of C code which can make use of the flex-generated scanner to manipulate input files. This section contains the logic of your program and will contain the largest part of the processing capacity of the resulting scanner.

The resulting lex.yy.c file is large relative to the the flex input file. A simple example flex input file is that listed above, namely:

```
%%
```

If you run flex on this, you get the file:

```
# include "stdio.h"
# define U(x) x
# define NLSTATE yyprevious=YYNEWLINE
# define BEGIN yybgin = yysvec + 1 +
# define INITIAL 0
# define YYLERR yysvec
...
struct yysvf *yyestate;
extern struct yysvf yysvec[], *yybgin;
# define YYNEWLINE 10
yylex(){
int nstr; extern int yyprevious;
while((nstr = yylook()) >= 0)
yyfussy: switch(nstr){
case 0:
if(yywrap()) return(0); break;
case -1:
break;
default:
fprintf(yyout,"bad switch yylook %d",nstr);
} return(0); }
/* end of yylex */
int yyvstop[] = {
0,
0};
# define YYTYPE char
struct yywork { YYTYPE verify, advance; } yycrank[] = {
```

```
0,0,    0,0,0,0,0,0,
0,0};
struct yysvf yysvec[] = {
0,      0,0,
...
yyunput(c)
int c; {
        unput(c);
        }
```

The ellipses (...) represent large pieces of missing code. In the interests of space, several hundred lines of code have been removed. The thing to realize is the complexity and size of this file. The power of flex is that it generates much of the parsing logic that you would have to write if you were to generate these routines yourself. You can affect the size and speed of the parsing by using the -C option. You can use a command like -Cem and generate the smallest table and, therefore, the slowest execution, or you can use the -CF option to generate the largest table and the fastest execution. This entire issue is related to tricks that flex does to increase parsing speed.

flex builds equivalence classes which are merely equivalent representations for patterns. For example, you can use the regular expression [a-c] which is equivalent to a b c. flex creates tables of information which store these equivalencies. The more equivalencies you store, the larger the table but the faster the processing time. This is the trade-off relative to the -C option.

Start conditions allow you to define conditional rules. To create a rule with a start condition, use the syntax:

```
<string>regexp
```

where <string> is any string which represents that start condition.
regexp is any regular expression.

For example, you can create a start condition known as KEVIN with a command like:

```
<KEVIN>"kevin"
```

This rule will be activated when you place the scanner in KEVIN start condition with a command like:

```
%s KEVIN
```

in the definitions section of the flex input file. You can also use the keyword BEGIN to define a start condition within your code. The typical implementation is to use a global boolean variable and test its value to define a start condition.

For example, you might see a section of code like:

```
if (name="KEVIN") then
        BEGIN(KEVIN)
etc....
```

This will execute the start condition KEVIN for the scanner.

There is a very good section on performance in the flexdoc.1 document included with flex. See this for more details on how to get the maximum performance out of your flex system.

There are a variety of macros which you can redefine with flex. The most basic are:

YY_DECL	Declaration of yylex
YY_INPUT	Redefines the way yylex gets it input
YY_USER_ACTION	Defines a user action to be executed before an action
YY_USR_INIT	Defines a user action to be executed before scan begins
yywrap()	Redefines the end of file condition return code

7.6.3 Examples

One of the simple ways you can use flex to manipulate text is to let the scanner remove any specified text strings from the input file. An example rules file named example.flex might look like:

```
%%
"string to remove"
```

You can first run flex on this file with the command:

```
$ flex example.flex
```

This generates a file lex.yy.c, which is the scanner file. This is a C code file which can be compiled and executed. Note that lex.yy.c does not have a main function declaration and therefore must be compiled and linked with the command:

```
$ cc -lfl lex.yy.c
```

This generates a file a.out which can then be executed as you would any other executable. Note that if you use lex (and not flex), you need to use the option -ll instead of -lfl to generate the same resulting executable.

Assume you have an input file named example.input which contains the following information:

```
line1
line2
line3
line4
string to remove
line 6
```

When you execute the command:

```
$ a.out < example.input
```

the resulting output will be:

```
line1
line2
line3
line4
line 6
```

You can redirect this as you normally would with the command:

```
$ a.out < example.input > example.output
```

Now the file example.output contains the above output.

7.6.4 Differences between lex and flex

The basic differences documented between flex and lex are:

yylineno lex variable is not supported by flex.

input() is not redefinable in flex, at least in accordance with POSIX specifications.

output() is not supported.

flex supports exclusive start conditions.

flex expands definitions with enclosing parens, while standard lex does not.

%r (ratfor) is not supported by flex.

After a call to unput(), yytext and yyleng are undefined.

The precedence of {} and ^ are different with flex.

To reference yytext outside of the scanner, you declare yytext as "extern char *yytext" in flex instead of "extern char yytext[]".

yyin is not defined until the first scanner execution with flex, unlike lex which defines yyin to stdin.

Special table-size declarations required by lex are not needed by flex and are ignored.

Multiple actions on the same line are supported by flex.

yyrestart() is available in flex.

yyterminate() is available in flex.

There are other subtle differences and most are documented in flexdoc.1. See this document if you need more details. Certainly performance is better with flex than with most lex implementations. As your rules files get large, this becomes a significant benefit.

7.6.5 Special flex 2.4.6 notes

With flex 2.4.6 there is a subdirectory named MISC which contains special files and notes for various platforms. See the README file for details on some new features of flex 2.4.6 as well as for a listing of obsolete features.

7.6.6 Installation

The installation of flex is very straightforward. There is a very basic makefile which contains a few comments at the beginning that discuss POSIX and System V support issues. By including a macro definition on the make command line, you can override any possible definitions, and the make should run successfully.

The build of flex is very straightforward. Issue the commands:

```
$ ./configure
$ make
```

This will create an executable named flex and will copy it to the /usr/local directory. There is also a file named makefile.rs6k which you can use in lieu of the configure command to use this type:

```
$ make cc="gcc" -f makefile.rs6k
```

To ensure that the build process occurred correctly, issue the command:

```
$ make test
```

There should be no results from the diff command. If there are, you have a problem and you need to go back and review your makefile for potential problems on your system. If not, you can install in the /usr/local default directories with the command:

```
$ make install prefix=/usr/local
```

if you want; otherwise, simply place the resulting flex executable in your path and have at it.

You may also want to modify the macros BINDIR, LIBDIR, AUXDIR, and MANDIR to point to some location other than the default /usr/lo-

cal. Any choice should be consistent with your product support methdology.

7.6.7 Conclusions

flex is a very powerful tool which provides functions above and beyond most vendor implementations of lex. One of its most powerful features is in its relationship to yacc, which was not discussed in this section. See the documentation accompanying flex for more details. There are several flex input files in the software on the accompanying CD. See the groff distribution for a good example of a lex/flex file and of a yacc/bison file.

Much of the information in the section was taken from the files flex.1 and flexdoc.1. flexdoc.1 contains a lengthy and comprehensive overview of flex and is a must read before attempting any real use of flex. Several of the examples in this section were taken directly from these documents, and they will prove invaluable as you become more proficient with flex.

7.7 bison

7.7.1 Introduction

bison is the GNU replacement for yacc (yet another compiler compiler). The standard input file has a standard suffix of .y and the output is C source code. bison is really a language and precompiler system which allows you to parse and modify input according to formats specified in the bison input file.

Bison is backward compatible with yacc and supports yacc input files. It is a GNU product and as such is subject to its GNU General Public License as included both in the product distribution and in App. C of this book.

7.7.2 Installation

The basic installation of bison is straightforward. bison uses the configure subsystem from GNU which constructs the appropriate Makefile from some configuration files.

As with most of the other newer GNU packages, you must first configure the software for your particular machine with the configure command:

```
$ configure rs6000-ibm-aix --prefix=/usr/local
--exec-prefix=/usr/local
```

This will generate the appropriate Makefiles for you to then build the product. If you have trouble with the configure command, see Sec. 7.4 for more information.

It appears that the bison Makefile assumes that you have the proper subdirectories created before it can correctly create files in them. This works well for /usr/local; however, if you change the path of the final executables with the --prefix, you need to manually create the following directories in the /usr/local directory before executing the make command: bin, man/man1, info, and lib. Once you have done this and configured your machine, you can build the software. To do this, use a command like:

```
$ make clean
$ make install
```

This will generate all executables and place them in lib, bin, and man directories under the /usr/local directory. If you don't specify the install option, the executables and other files will be copied into the directories specified with the prefix option on the configure command. Using the install option is not a recommended practice in large environments since you may unintentionally overwrite software. See Sec. 6.6 for more information.

You can override macros and options with either configure or make. See Sec. 7.4 for more information.

Once you have compiled and linked bison, you are ready to execute as described in the rest of this chapter. If you have questions or needs, reference the INSTALL file in the bison distribution; it contains more information specific to configure and make which may help with the bison installation.

7.7.3 Usage

The basic syntax for bison is:

```
bison [-b file-prefix] [--file-prefix=file-prefix] [-d] [--defines]
[-l] [--no-lines] [-o outfile] [--output-file=output-file]
[-p prefix] [--name-prefix=prefix] [-t] [--debug] [-v] [--verbose]
[-V] [--version] [-y] [--yacc] [--fixed-output-files] file
```

where -b file-prefix defines a prefix to be used by all bison output files.
--file-prefix=file-prefix is the same as -b file-prefix.
-d generates an additional output file containing macro definitions for the token type names defined in the grammar and the semantic value type YYSTYPE. This file has an exension of .h and a filename which is the same as file.

--defines is the same as -d.
-l does not place #lines in the bison output file.
--no-lines is the same as -l.
-o outfile specifies output filename.
--output-file=outfile is the same as -o outfile.
-p prefix renames external symbols used by the parser to use prefix instead of the default yy.
--name-prefix=prefix is the same as prefix.
-t outputs YYDEBUG so debugging is enabled.
--debug is the same as -t.
-v is the verbose option. The output file containing the verbose information has the same filename as the input file with an extension of .output instead of .tab.c.
--verbose is same as -v.
-V displays the version of bison.
--version is the same as -V.
-y makes output filename y.tab.c to maintain backward compatibility with yacc.
--yacc is same as -y.
--fixed-output-files is same as -y.
file is the bison input file to be operated on.

As you can see, most commands have two options that accomplish the same thing. This is historical and has to do with different parsing mechanisms on UNIX and how you might establish your input parsing. There should be no differences between the single and double hyphen options to the bison command.

See the included bison.1 manual page and bison.texinfo file for more information since much of this information was taken directly from there.

7.7.4 Differences between yacc and bison

bison is very compatible with yacc, including support for yacc input files. The only differences are in the output naming conventions and the @N operator.

bison will create an output file with the same filename as the input file except for changing the .y input file suffix to a .c. In other words, yacc always creates an output file named y.tab.c regardless of the input filename. bison, on the other hand, will keep the original filename. If you have a bison input file named kevin.y, when you execute bison, you will get an output filename of kevin.tab.c. This is a nice advantage for bison, but it does tend to confuse make since it has default rules about

yacc output filenames. Keep this in mind when you are building your software systems.

The other key differences are in bison's support of @N. The @N gives you access to the character number, beginning line number, and ending line number of symbols in the current rule. This is not available with yacc.

7.7.5 Differences between yacc and yacc

There are several versions of yacc available, including a Berkeley version and an AT&T version, which provide slightly different functionality. You need to be extremely careful as to which one you are using because you may or may not get different behavior. This is dependent not only on which version of UNIX you are using but on your path on a given machine. For example, on some versions of SunOS, you have two versions of yacc, one in /usr/bin and the other in /usr/5bin. Keep this in mind when building makefiles and developing software systems.

The differences between tools on different UNIX platforms are one of the main reasons to use a tool like bison; it buys you complete portability between heterogeneous platforms. This becomes essential as you distribute your software systems to more than one UNIX architecture.

7.7.6 Conclusion

bison is a parser generator which makes it easy to parse and interpret command input lines as well as file contents of a given structure. Parsing is one of the most time consuming of all software development activities, and bison provides you with a faster interface and developent cycle for generating this parsing code.

7.8 patch

7.8.1 Introduction

patch is a GNU application which is responsible for providing and installing patches to GNU software systems. patch supports the various outputs from diff as well as the diff output from the GNU diff executable.

By simply distributing a diff file, you can use patch to generate any required changes from one version to the next. An example of that is included on the accompanying CD with Ghostscript. If you look in the tar subdirectory for Ghostscript, you will see four compressed patch files. Uncompress (gzip -d) them and examine them to learn more about the structure of the patch input files.

7.8.2 Usage

The basic syntax for the patch command is:

```
patch [-b suff] [-B pref] [-c] [-d dir] [-Eefsv] [-o file] [origfile
[patchfile]] [+ [options][origfile]]...
```

where -b suff—-suff is the backup extension instead of .orig or N.
-B pref.—pref is the prefix of the backup filename.
-C forces patch to use patch file as a context diff.
-d dir changes directory to dir before executing any patch commands.
-D sym causes patch to use #ifdef... #endif construct to make changes.
-E removes empty files after patches are applied.
-e causes patch to treat patch file as an ed script.
-f forces mode for patch.
-l is loop patch. This ignores whitespace mismatch problems in the diff.
-s is silent mode.
-n interprets patch as a normal diff.
-t is similar to -f but skips patches for which a file to patch can't be found and those with mismatched versions.
-v displays patch version.
-o file generates output file named file.
origfile is original file to be patched.
patchfile is file containing patch information.

There are many other options which allow you to change the structure of your patch files and change the default behavior of the patch command; however, since you will probably never use them, they are not documented here. For exact information on the syntax of the patch command and all associated options, see the manual page patch.man in the main patch directory.

As stated in the introduction, patch applies the results of diff commands to files. This provides a simple and consistent way of providing software patches to systems currently in use. patch is the most common way Internet software is patched and is therefore widely used.

The default behavior of patch is to apply the diffs to a given file and replace the original with the updated file. The original file is given an extension of .orig in the same directory. If the backup already exists, patch creates a copy of the backup file by changing the first lowercase letter in the file extension to uppercase. If all characters in the backup filename are uppercase, it begins removing a letter from the name until it creates a unique filename. This is then the second-level backup file.

patch supports several different types of patch files and is intelligent enough to determine which type of patch file it is operating on. You can override this with command line options, but this is not recommended unless you know what you are doing. patch also has some intelligence to help it determine when a patch files is corrupted, and it attempts to fix the patch file to ensure the correct updates to the original files. If it cannot, you will get an error, and patch will fail.

If you are patching large files, you may want to set the TMPDIR variable which specifies the location of all temporary files. The default location is /tmp. Finally, you can set the variable SIMPLE_BACKUP_SUFFIX to change the default backup file suffix from .orig to anything you want.

The most common use of patch is without command line arguments and without origfiles. The basic syntax is:

```
$ patch <file
```

where file is the patchfile. This will apply all patches within file to all corresponding files in the current directory, creating new original files and renaming the original file with a .orig file extension.

Ghostscript 2.6.1 on the accompanying CD contains four patches which are contained in the tar subdirectory for Ghostscript itself. A piece of one of the patches is included here to illustrate the structure of a patch file:

```
Ghostscript 2.6.1 Patch #1
```

To apply this patch, cd to the directory containing the ghostscript source and use:

```
patch -s < ThisFile
```

patch will work silently unless an error occurs. If you want to watch patch do its thing, leave out the "-s" argument to patch.

See the readme.fix file, which follows, for a summary of the fixes:

```
*** /dev/null Sun Jun 27 07:26:01 1993
--- readme.fix Thu Jun 17 11:18:48 1993
***************
*** 0 ****
--- 1,118 ----
+ Copyright (C) 1993 Aladdin Enterprises. All rights reserved.
+
+ This file is part of Ghostscript; it is licensed under the same
+ terms as the rest of Ghostscript. If you do not have Ghostscript,
+ you do not have the right to have this file.
+
+ Fixes for Ghostscript 2.6.1
```

```
+ ---------------------------
+
+ (last update: 6/14/93)
+
+ This file summarizes a number of important quality fixes for
+ Ghostscript 2.6.1. The fixes are supplied in the form of
+ replacements for corresponding files in the 2.6.1 release. Please
+ report any problems.
+
+ 6/5/93
+ ------
+
+ Problem:
+  The Unix install script used gs rather than $(GS) as the name of
+  the executable.
+  The Unix install script didn't copy gs_dbt_e.ps to $(gsdatadir).
+ Files affected:
+  unixtail.mak (and unix-*.mak, built from it using tar_cat)
+
+ Problem:
+  The ps2ascii script still referenced ps2ascii.ps under its
+  old name gs_2asc.ps.
+ Files affected:
+  ps2ascii
+
+ Problem:
+  ps2image.ps had a 'pop' missing in the written-out definition of
+  'max' in the boilerplate code it put at the beginning of
+  compressed files.
+  ps2image.ps got a typecheck if a scan line had no repeated
+  data in it anywhere.
+ Files affected:
+  ps2image.ps
+
+ Problem:
+  rectfill drew rectangles with vertices specified in clockwise
+  order as 0-width lines.
+ Files affected:
+  gsdps1.c
+  gdevx.c
...
*** 1.1 1993/06/27 12:24:14
--- devs.mak 1993/06/09 08:27:08
***************
*** 108,113 ****
--- 108,114 ----
  # gifmono Monochrome GIF file format
  # gif8 8-bit color GIF file format
  # pcxmono Monochrome PCX file format
+ # pcxgray 8-bit gray scale PCX file format
  # pcx16 Older color PCX file format (EGA/VGA, 16-color)
  # pcx256 Newer color PCX file format (256-color)
  # pbm Portable Bitmap (plain format)
***************
*** 756,765 ****

pcx_=gdevpcx.$(OBJ) gdevpccm.$(OBJ) gdevprn.$(OBJ)

! gdevpcx.$(OBJ): gdevpcx.c $(PDEVH) $(gdevpccm_h)

pcxmono.dev: $(pcx_)
 $(SHP)gssetdev pcxmono $(pcx_)
```

```
pcx16.dev: $(pcx_)
 $(SHP)gssetdev pcx16 $(pcx_)
--- 757,769 ----

pcx_=gdevpcx.$(OBJ) gdevpccm.$(OBJ) gdevprn.$(OBJ)

! gdevpcx.$(OBJ): gdevpcx.c $(PDEVH) $(gdevpccm_h) $(gxlum_h)
pcxmono.dev: $(pcx_)
 $(SHP)gssetdev pcxmono $(pcx_)
+
+ pcxgray.dev: $(pcx_)
+  $(SHP)gssetdev pcxgray $(pcx_)
pcx16.dev: $(pcx_)
 $(SHP)gssetdev pcx16 $(pcx_)?
*** 1.1 1993/06/27 12:24:14
--- gdevpcx.c 1993/06/01 12:21:26
***************
*** 1,4 ****
! /* Copyright (C) 1992 Aladdin Enterprises. All rights reserved.
This file is part of Ghostscript.
--- 1,4 ----
! /* Copyright (C) 1992, 1993 Aladdin Enterprises. All rights
   reserved.
This file is part of Ghostscript.
***************
*** 20,25 ****
--- 20,26 ----
 /* PCX file format devices for Ghostscript */
 #include "gdevprn.h"
 #include "gdevpccm.h"
+ #include "gxlum.h"
/* Thanks to Phil Conrad for donating the original version */
/* of these drivers to Aladdin Enterprises. */
***************
```

The first part of the above patch contains documentation which is inserted into the file readme.fix. This is documented by the section:

```
*** /dev/null Sun Jun 27 07:26:01 1993
--- readme.fix Thu Jun 17 11:18:48 1993
***************
*** 0 ****
--- 1,118 ----
```

The original file to be modified is surrounded by asterisks (*), while the modified file information is surrounded by hyphens (-). In this case there is no original file, and a new file readme.fix is created with 118 lines (denoted by 1,118.)

The second file to be modified is devs.mak. The section:

```
*** 1.1 1993/06/27 12:24:14
--- devs.mak 1993/06/09 08:27:08
***************
*** 108,113 ****
--- 108,114 ----
```

says that devs.mak original lines 108 through 113 should be replaced by lines 108 through 114, which are documented directly below the --- 108, 114 ---- line.

This is standard output structure from a diff command. See the diff manual page for more information and create some simple examples to understand more clearly how this patch system works. However, unless you are going to create patches, it will probably be unnecessary for you to understand the structure of the patch files to use them.

Most often the syntax is:

```
$ patch < patchfile
```

in the proper directory and the rest is magic.

7.8.3 Installation

The installation of patch is very simple. patch uses the configure system to create a relatively straightforward makefile which can then be executed normally. The basic commands are:

```
$ ./configure prefix=/usr/local
$ make
```

You can also use the makefile provided on the CD with the command:

```
$ make cc="gcc" -f makefile.rs6k
```

Note that the configure is intelligent enough to generate the appropriate makefile, and the makefile works without any additional option. If you want to place the binaries in some other directory than the main patch subdirectory, use the make install option (for more details, see Sec. 7.4).

There were no problems with this simple methodology on the RS/6000 running AIX. You may have to change a compilation or linking variable if you encounter problems on your machine, but it is unlikely.

7.8.4 Conclusion

patch is a very powerful tool which allows you to distribute software updates easily and cleanly. It is also the way most Internet tools are now updated and should be another tool in your personal UNIX tool kit.

By supporting both standard diff and GNU diff, patch allows diff files to be used to support distributed file maintenance and updates.

7.9 gas

7.9.1 Introduction

gas is the GNU assembler. It is an assembler which compiles and creates binary files from assembler code. It also provides a cross-compiler capability for assemblers. This means that you can compile assembler code to run on an architecture machine that is different from the one on which it is compiled. This is a powerful feature and is often used by software engineers to support heterogeneous platforms as easily as possible.

Even though it can be used as a stand-alone assembler package, its primary function is to serve as an assembler for many of the other GNU packages. While there are assemblers distributed with vendor systems, gas is a very sophisticated package which supports all of the kind of things GNU products tend to do. You may want to reference this and install it before you begin to install and configure most of the other GNU products described in this book.

gas is a GNU product and as such is subject to its GNU General Public License as included both in the product distribution and in App. C of this book.

7.9.2 Usage

The basic syntax of gas is:

```
as [-a|-al|-as] [-D] [-f] [-Idir] [-K] [-L] [-ofile] [-Rvw]
[-- | file ...]
```

where -a creates an assembly listing file including symbols.
-al creates assembly listing only.
-as creates symbol listing only.
-D is ignored by as; used for as compatibility.
-f does not perform any preprocessing.
-Idir uses dir for include files.
-k issues warnings when differences tables are altered for long displacements.
-L keeps local symbols.
-ofile—output file is file.
-R merges data section into text section.
-v displays as version.
-w supresses warning messages.
-- is standard input.
file... is one or more files to assemble.

Much of the above information was taken from the as man page named ./gas/doc/as.1 included in the distribution on the accompanying CD. See this file for more information.

There is also information in texinfo format which can be viewed with either texinfo mode in emacs or with the info program available with the texinfo distribution on the CD. There is also a section on documentation in the ./gas/README file in this distribution. If you want to create TeX output files, you will need to install TeX on your machine.

If you want to build the TeX.dvi (device independent) files, you must use a command like:

```
$ cd gas-2.1.1/gas/doc
$ make as.dvi
```

and have TeX installed and available. If you want to build the texinfo files, use a command like:

```
$ make info
```

instead of the make as.dvi as listed above. Note that here you must have the texinfo available. See Sec. 8.3 for more information.

7.9.3 Installation

The installation of gas is very straightforward and consists of moving to the main directory of the gas distribution and typing:

```
$ limit mem u
$ limit data u
$ ./configure
```

The first two commands ensure that you have enough memory and data space available. If you need to know your current target, type the command:

```
$ ./config.guess
```

This will generate a target string for you. For example, on AIX 3.2 with software in the /usr/local/gas/gas-2.2.3 subdirectory you might type:

```
$ ./configure --prefix=/usr/local/gas
```

Then you simply issue the make command:

```
$make CC=gcc
```

This will build all the proper executables and libraries for gas. There are a few additional notes in the NEWS file and the associated README files in the gas subdirectory. See these if you want more information. Of particular interest is the file README.coff, which de-

scribes support for coff and "vanilla" linkers. See this if you are interested in coff support.

If you would then like to install the package in the /usr/local/bin area, simply type:

```
$ make install
```

7.9.4 Conclusion

gas provides an assembler for many machines and tools, including gcc, gdb, g++, emacs, and many others. If you use gas, you will be assured of minimal problems when building these tools. You can also use gas as a very powerful assembler which supports multiple architectures and machines. Take a look at this technology if you are interested in assembler technology since it is a fairly good implementation of a cross-platform assembler.

7.10 gdb

7.10.1 Introduction

gdb is the GNU debugger which you can use to interactively debug executable6 applications in a UNIX environment. gdb works much like dbx. It provides interactive debugging capabilities for languages such as C, and C++ and also supports remote debugging and cross-debugging capabilities that allow you to debug a remote application on a hardware architecture that is different from the local one. The interface for remote debugging is still somewhat primitive and will require some effort to configure. There are stubs which can be modfied, including m68k-stub.c , i386-stub.c, and sparc-stub.c. Examine these for more details on how to perform remote debugging over a serial line.

gdb is a GNU product and as such is subject to its GNU General Public License as included both in the product distribution and in App. C of this book.

7.10.2 Usage

The basic syntax of the gdb command is:

```
gdb [-help] [-nx] [-q] [-batch] [-cd=dir] [-f] [-b bps] [-tty=dev]
[-s file] [-e prog] [-se prog] [-x file] [-d-dir] [-c file]
[-c file] [prog [core | ID]]
```

where -help displays interactive help screens.
-nx doesn't run any initialization files such as .gdbinit.
--q is quiet mode. Doesn't print introductory message.

-batch executes all commands given by the -x option and returns.
-cd=dir uses dir as the working directory.
-f is used with emacs to output current position information within the file.
-b bps sets line speed of any serial line used for remote debugging.
-tty=dev defines dev as standard input and output.
-s file reads symbol table from file.
-e prog uses prog as the executable file.
-se prog uses prog as the executable file and reads the symbol table from it.
-x file is file containing gdb interactive commands to be executed.
-d=dir searches dir for executable files.
-c file uses file as a core file to examine.
prog is executable to run with gdb.
core is the core dump to analyze.
ID specifies the id of the process you want to attach to.

gdb gives you the ability to debug executable programs, analyze core dumps, and attach to running processes and analyze their operation. As with all interactive debuggers, gdb provides basic capabilities to monitor variables and program operation and alter execution by changing values and conditions during execution.

Basic interactive operation. The basic commands available to gdb are very similar to dbx. Some of the basic interactive commands are:

break	next
bt	quit
continue	run
help	step
print	

These are all documented both in the manual page shipped with gdb and the Postscript reference card refcard.ps. See these for more documentation and more commands for gdb. Also see Sec. 4.2 for more information on debugging techniqes and topics.

There continue to be many enhancements to the gdb product, most of which are documented in the file ./gdb/NEWS. See this file for changes listed by release to gdb for the last few revisions. The other file of inter-

est is the ./gdb/README file which documents many of the issues you will encounter when building gdb and looking at the documentation.

A prebuilt info file (for the Info system and emacs) is contained in this release as ./gdb/gdb.info. Use emacs or some other tool to examine this. See Sec. 8.3 for more information on this. There is also a new file named gdb-4.13/gdb/refcard.ps which is a Postscript file containing a nice reference card on all gdb commands and syntax. You can build this file if you need to from the texinfo input file with the command:

```
$ make refcard.dvi
```

Note that you need TeX to build this file. See the README file for more details.

7.10.3 Installation

The installation of gdb is very simple. First uncompress the distribution file with gzip -d or gunzip. Next unwind the tar file into a local software directory. The directory structure created begins with a root directory of gdb-4.13 and includes several subdirectories, including library directories, help directories, a texinfo subdirectory, and a README file.

To build gdb, use the commands:

```
$ configure TARGET --prefix=dir
```

where TARGET is the architecture you are currently using and the dir is the prefix where you would like the resulting files placed. If you have questions about the configure and build process, see Sec. 7.4.

A simple example of building on an AIX machine might be:

```
$ ./configure rs6000 --prefix=/usr/local

$ make CC=gcc
```

If you using gmake, you might multithread the build with a command like:

```
$ make -j5 CC=gcc
```

This creates the gdb executable in the gdb subdirectory. You can move this into any directory according to your product structure specifications. You can create different versions of gdb for different target machines based on the --target option in the configure command. This gives you the ability to debug a program on a remote machine that is not of the same architecture as the local machine. See Sec. 7.4 and the

gdb/README file in the gdb subdirectory for more information on building cross-compiling capabilities with configure and gdb.

There is a premade reference card in a file named ./gdb/refcard.ps which is a Postscript file and can be printed to any Postscript printer. You can also preview this document and view it on line with a tool like Ghostscript. You will make use of this quite often until you get more familiar with the gdb tool. There is TeX-formatted output in the gdb.info file which can be built and used by emacs, and other tools can be used to manipulate and view this information.

Finally, there is a gdb test suite which requires the presence of a tool called dejagnu, which is available from any Internet source. If you are interested in this, see the ./gdb/README file for more information.

Known problems. There are several known problems, including problems with backtraces on the RS/6000, incorrect reporting of struct values on SunOS, and breakpoint problems when watchpoints are enabled. See the gdb/README file for more details.

Examples. To debug a file named kevin.out, use the command:

```
$ gdb kevin.out
```

To analyze a core file and determine where the crash occurred, use the command:

```
$ gdb core
```

You can also attach to a running process with a command like:

```
$ gdb kevin.out 1111
```

which will attach to a process with an id of 1111. You can attach to and detach from a number of processes during a gdb session. See on-line help for more information. See also Sec. 4.2 for more details on similar debugging techniques.

7.10.4 Conclusion

gdb is a very powerful debugger which provides most of the capability you would want in an interactive debugger. It also works well with other GNU products such as gcc and gas and allows for advanced techniques such as cross-compilation and cross-debugging.

For more information than is available in this book and the delivered documentation, see *Using GDB: A Guide to the GNU Source Level De-*

bugger, by Richard Stallman and Roland Pesch, available from the Free Software Foundation.

7.11 gawk

7.11.1 Introduction

gawk is an acronym for GNU awk. It is upwardly compatible with SVR4's awk and supports nawk and awk features. The release that this chapter is based on is gawk 2.15.4. With gawk, you can run awk on almost every UNIX platform and ensure that you are running the same awk on every platform; therefore, the performance will be the same. gawk supports awk, as described in the book *The AWK Programming Language* by Aho, Kernighan, and Weinberger, and some GNU-specific extensions.

One of the problems with awk on different platforms has been inconsistencies in the performance of this tool on different flavors of UNIX. With gawk as with other GNU products, you eliminate the inconsistencies and provide a stable awk development platform for most versions of UNIX.

With gawk 2.11 and beyond, there is full DOS support, so you can begin to take your gawk scripts from UNIX to DOS without portability problems.

For a more complete discussion of the awk syntax, see Sec. 5.2, which contains a variety of examples and more discussion of the awk language itself. This is directly applicable to gawk and in fact your use of gawk or awk should be transparent.

gawk is a GNU product and as such is subject to its GNU General Public License as included both in the product distribution and in App. C of this book.

7.11.2 Installation

All the files in the tar archive will be unwound and placed in directories under your current working directory. You are now ready to examine files and build the product for your platform. Note that this process is unnecessary for the files on the accompanying CD since they are already unwound; however, you may need to follow this procedure if you get a different version from the Internet or some other source.

Files. As is standard with most Internet products, the first file to look for is the README file. Once you find this, print it out and examine it in detail. It contains information on the current release features and functionality as well as on bugs and installation notes.

Along with the README file, there are two files, FUTURES and

PROBLEMS, which describe future enhancements and features to the product and known problems with the current gawk. Consult these two files for more information about relevant topics.

The other file of immmediate interest is the Makefile. As is standard with most Internet software products, the Makefile is tailored for the Sun environment. gawk also has built in some dependencies on gcc (GNU's C compiler; see Sec. 7.2 for more information) which you must watch out for. Different C compilers support different functions within C. This is discussed in more detail in Sec. 7.2; however, suffice it to say that you will need to change switches on the make compiles for different C compilers to work correctly.

Release 2.15.4 of gawk includes a simple version of the GNU configure tool which supports only the architecture option and none of the --options that most other tools use. You need to look in the config directory for the appropriate file. In this case it is rs6000. Therefore to configure for this, type:

```
$ configure rs6000
```

To see what configurations are supported, simply type:

```
$ configure
```

This will list all supported machine types.

In Makefile.in, there are sections outlined by comments which tell you which platforms are supported with which commands in the Makefile. Because of the changing standards in the C language, several flavors are shown. If you are running a compiler which is ANSI compliant, you should select the ANSI sections of the Makefile and make sure that the ANSI-compliant switch on your C compiler is invoked by the Makefile. If this is a problem, consult your C compiler documentation for more detail. After configuring as explained above, issue the following command to build the product:

```
$ make
```

You can rebuild from scratch if you would like by issuing the commands:

```
$ make clean
$ make
```

Next you need to install it; use the command:

```
$ make -n install
```

to verify where it will be installed. If this is not appropriate, change the makefile and the insure command:

```
$ make install
```

The make clean command cleans out any previously existing executables. This ensures you of a new set of executables specifically for your environment. See Sec. 7.5 for more details on this process.

There is a file named makefile.rs6k included on the CD. Using this, you can type;

```
$ make cc=gcc -f makefile.rs6k
```

There are machine-specific README files for a variety of machines. For example, there is a file README.rs6000 which documents an issue with the alloca routine. For the most part you can ignore this for the build.

Finally you will find documentation in the form of gawk.1 files in the directories. The file extension .1 normally means that the file is a help file and comes from section 1 of the UNIX documentation, which contains general user commands. You can access and print this file to the screen or a printer with the nroff commands. For example:

```
$ nroff -man gawk.1 | more /* print out manual page to the screen */
```

or

```
$ nroff -man gawk.1 | lpr /* prints out command to default printer */
```

Printing out the manual page for future reference is one of the best ways to use Internet software products and will allow you to understand a product's exact functions relatively quickly. See Sec. 6.10 and other sections in Chap. 6 for more information. Other documentation is supplied in TeX format. TeX is a macro language which provides very sophisticated command-driven word processing and typsetting capabilities.

There are a variety of sources listed in the README file for bug reports and fixes as well as general information for the gawk product. See these for more details on the gawk product.

7.11.3 Usage

gawk can use either POSIX-style commands, which are preceded by a single hyphen, or GNU-style options, which are preceded by two hyphens. To simplify the documentation, we will use the POSIX-style

syntax. See the manual page (gawk.1) if you want more information on the GNU syntax.

The syntax for the gawk command is:

```
gawk [-a] [-e] [-W compat] [-W copyleft] [-W help] [-W lint]
[-W posix] [-V] [-Fsep] [-v var=value] -f prog [--] file ...

gawk [-a] [-e] [-W compat] [-W copyleft] [-W help] [-W lint]
[-W posix] [-V] [-Fsep] [-v var=value] [--] progtext ...
```

where -a uses awk-style regular expressions. This is the default.
-e uses egrep expressions as documented in the POSIX standard.
-W compat is the compatibility mode. gawk runs just like awk.
-W copyleft prints GNU copyright.
-W help displays options.
-W lint runs lint to check potential programming problems.
-W posix enables compatibility mode with awk.
-V prints current version of awk to standard error.
-Fsep defines field separator.
-v var=value defines BEGIN block variables.
-f prog is awk command input file.
-- signals the end of options. Used to separate options from files.
progtext—awk commands are included in line instead of in a file prog as with the -f switch.

gawk looks very similar to awk in terms of syntax and command structure. There are some things to point out, however. There continue to be changes in gawk, many of which are documented in the NEWS file. See this for more information on gawk functionality.

When searching for files to be used with the -f switch, the AWKPATH variable is examined to detmine fully qualified paths. The default if no AWKPATH is defined is .:/usr/lib/awk:/usr/local/lib/awk.

As mentioned earlier, gawk is compatible with SVR4 awk and supports features such as preexecution variable definition with the -v var=val, multiple -f option concatenation, ANSI C printf support, and the \a, \v, and \x escape sequences. See the README file for more information.

Finally, GNU added some extensions which are unique to gawk such as:

IGNORECASE variable which allows for character case independence.

AWKPATH variable.

-a, -e, -c, -C, and -V options.

7.11.4 Conclusion

gawk is a powerful tool which not only provides awk and nawk compatibility but also provides its own set of features and functions that add value to the original awk functionality. It also runs on multiple platforms and provides a stable and uniform environment for developing and running awk scripts.

7.12 RCS

7.12.1 Introduction

RCS stands for Revision Control System. The version discussed in this chapter is 5.6.01. RCS is written and maintained by Walter F. Tichy at Purdue University. GNU distributes it along with their MANIFEST, which describes the copyleft licencing arrangement. Because of this, you can freely distribute RCS and thus it is included in this book.

RCS provides function similar to SCCS in that it provides version control and configuration management. By providing a series of commands which control and track source files, RCS provides you with the ability to better control and monitor your software development processes. This section is by no means an exhaustive overview of the capabilities and subtleties of RCS. Consult the documentation distributed with RCS for more information. There are manual pages in the man subdirectory and Postscript documents (*.ps) in the doc subdirectory. Specifically, there is a document titled rcs.ps which contains a very good overview and description of RCS and version control.

RCS is a GNU product and as such is subject to its GNU General Public License as included both in the product distribution and in App. C of this book.

7.12.2 Usage

The basic usage of RCS is included in the following commands:

`ci`	Checks in revisions
`co`	Checks out revisions
`ident`	Extracts identification markers
`rcs`	Changes RCS file attributes
`rcsclean`	Cleans working directory
`rcsdiff`	Compares revisions
`rcsfreeze`	Freezes a configuration
`rcsmerge`	Merges revisions
`rlog`	Reads log messages

Each command is discussed in more detail below.

Checking in a file. The basic syntax is:

```
ci [{-lrufkqIM} [rev]] [-ddate] [-mmsg] [-nname] [-Nname] [-sstate]
[-tfile] [-t-string] [-wlogin] [-Vn] [-xsuff] file ...
```

where -l[rev] checks in file(s) and automatically rechecks this file out and locks it for future editing.
-r[rev] checks in file(s) and releases the lock.
-u[rev] checks in file(s) and immediately rechecks this file out but doesn't lock it.
-f[rev] forces the file checkin even if the file hasn't changed.
-k[rev] searches the checkin file for information keywords containing author, change time, state, etc., to place in RCS instead of computing them locally upon checkin.
-q[rev] is quiet mode.
-I[rev] is interactive mode.
-M[rev] sets the modification time for checkin file.
-d[date] uses current date for checkin.
-msg uses string as the message for checkin log.
-nname assigns name to name of revision instead of simply using a number to represent the revision number.
-Nname is same as -nname except that it will override any previous definition of name.
-sstate sets state of checkin file(s).
-tfile writes text contained in file to RCS.
-t-string writes text contained in string to RCS.
-wlogin assigns login as owner of checkin rather than current userid.
-Vn emulates RCS version n instead of the current version.
-xsuff specifies information regarding the location of file(s) to be checked in.
file ... is one or more files specified to operate on.

ci is the command which checks files into RCS. You must either own or be the supervisor of the branch for that particular revision. You can override protections and locks with switches on the ci command line.

You control all checked in files with a concept called a revision. The revision tracks all file releases and relates them to other files. The default revision is 1.1 if no previous matching RCS file exists. ci works with files it calls working files. Working files are the files to be checked in, while the files stored in RCS are called RCS files.

Both the working file and RCS files can be specified explicitly. For example:

```
$ ci test.c
```

will create a file ./RCS/test.c and remove test.c. Placing the RCS file in the RCS subdirectory is the default action for ci. You can specify a resulting RCS filename explicitly. For example:

```
$ ci test.c testing/RCS/test.c
```

This takes the file test.c and checks it into RCS in the subdirectory ./testing/RCS as the filenamed test.c.

You can also specify file suffixes for the resulting RCS files. For example:

```
$ ci test.c testing/RCS/test.c,1
```

where ,1 is the suffix for the RCS file. You can check in a file named test.c in the current directory by merely specifying the RCS file. For example:

```
$ ci RCS/test.c,1
```

will look in the current working directory for a file named test.c and check it in. There are several other ways you can manipulate filenames. See the ci man page for more information.

Checking out a file. The basic syntax is:

```
co [{-lrufpqIM} [rev]] [-kkv] [-kkvl] [-kk] [-ko] [krev] [-kv]
[-ddate] [-sstate] [-w[login]] [-jlist] [-Vn] [-xsuff] file ...
```

where -l[rev] checks out file(s) and automatically locks for future editing.
-r[rev] retrieves latest file or one which matches rev.
-u[rev] retrieves the latest file or the one specified by rev unless the file is locked; then it unlocks and retrieves the file.
-f[rev] forces the working file overwrite.
-p[rev] displays retrieved revision on standard output instead of including in the file.
-q[rev] is quiet mode.
-I[rev] is interactive mode.
-M[rev] sets the modification time for the new working file.
-kkv generates keyword string based on default information. Username is not included unless the retrieved file is locked.
-kkvl is the same as -kkv except that username is always included.
-kk generates keyword strings without substitution.

-ko generates old keyword string as it existed before the file was checked in.
-k[rev] searches the checkin file for information keywords containing author, change time, state, etc., to place in RCS instead of computing them locally upon checkin.
-kv generates only keyword substitutions and not keyword strings.
-d[date] retrieves file whose date is equal to or later than that specified by date (date can be specified in free format; see the co man page for more information).
-sstate retrieves latest version whose state is state.
-w[login] checks out latest revision owned by login.
-jlist joins revisions together by taking first comma-separated pair from list and generates output passed to the next pair (see rcsmerge).
-Vn emulates RCS version n instead of the current version.
-xsuff specifies information regarding location of file(s) to be checked in.
file ... is one or more files specified to operate on.

There are a variety of keywords which are placed either in the check out file or displayed to standard output. Some of the most important are:

$Author$	Log
$Date$	$Revision$
$Header$	$Source$
$Locker$	$State$

Most of these keywords are self-explanatory. For more information and more keywords, see the co man page. Files are manipulated much the same as with ci. See the ci section above for more information.

Searching for keywords. The basic syntax is:

```
ident [-q] [file ...]
```

where -q is quiet mode.
file ... is one or more files to search.

ident searches all named files for all RCS keywords. These are designated by $string$. The result is displayed to standard output.

Changing RCS file attributes. The basic syntax is:

```
rcs [-ILUiq] [-alogins] [-Aoldfile] [-e[logins]] [-b[rev]]
[-cstring] [-kstring] [-l[rev]] [-u[rev]] [-Mn:log] [-nname[:[rev]]]
[-Nname[:[rev]]] [-orange] [-sstate[:rev]] [-t[file]] [-t-string]
[-Vn] [-xsuff] file ...
```

where -I is interactive mode.
-L sets strict checkin, which is useful for shared files; enforces locking even for the file's owner.
-U sets nonstrict checkin, which means that the owner of a file can check in and check out without locking restrictions.
-i creates and revises a new RCS file but doesn't deposit anything.
-q is quiet mode.
-alogins appends logins list (comma-separated usernames) to the access control list for a file.
-Aoldfile appends login list for oldfile to file.
-e[logins] removes named or all names on the access control list for file.
-b[rev] sets the default branch to rev or the highest if rev is not specified.
-cstring makes string the comment displayed before each Log display.
-kstring sets the default keyword substition to string.
-l[rev] locks the revision rev.
-u[rev] unlocks the revision rev.
-mn:log replaces revision n's log message with log.
-nname:[rev] sets the revision specified to the symbol name. If rev is not specified, the symbolic name is deleted.
-Nname:[rev] is the same as -nname but overrides any previously defined symbolic names.
-orange removes revisions specfied by range of the form rev1[:rev2], etc.
-sstate:[rev] sets revision rev to state where state is merely a symbolic name which represents the state of the revision.
-t[file] replaces contents of specified RCS file with file.
-t-string replaces descriptive log text of specified RCS file with string.
-Vn emulates RCS version n.
-xsuff uses suffixes (see the ci section above).
file ... is one or more files to operate on.

RCS either creates a new RCS system or modifies the attributes of an existing one. This is the first command you will execute to begin an RCS library for a software system.

Cleaning up working files. The basic syntax is:

```
rcsclean [-kstring] [-n[rev]] [-q[rev]] [-u[rev]] [-Vn] [-xsuff]
[file...]
```

where -kstring uses subst keyword substitution (see co for more details).
-n[rev] displays the results of the rcsclean command without actually executing it.
-q[rev] is quiet mode.
-u[rev] unlocks rev if no difference is found.
-Vn emulates RCS version n.
-xsuff uses suffixes to control RCS files.
file ... is file or files to be operated on.

rcsclean removes any files that have been checked in since they were last modified. You can either specify files to be checked and removed, or if no files are specified, all files in the subdirectory RCS are checked against working files in the current directory and removed.

Comparing revision differences. The basic syntax is:

```
rcsdiff [-kstring] [-q] [-rrev1 [-rrev2]] [-Vn] [-xsuff] [diff
options] file ...
```

where -kstring is keyword substitution (see co section).
-q is quiet mode.
-rrev1 -rrev2--if only rev1 is specified, its contents are compared with its working file; if both are specified, the corresponding files are compared between revisions; and if no revisions are specified, the latest revision is compared with its corresponding working files.
-Vn emulates RCS version n.
-xsuff uses suffix notation for RCS files.
diff options includes all options from the diff command.
file ... is one or more files to operate on.

rcsdiff generates diff commands on various specified revision files and/or working files outside of RCS. You can even specify diff options to the rcsdiff command. The output of this command looks much like that of the diff command and can be used with other commands to merge file changes.

Freezing configurations. The basic syntax is:

```
rcsfreeze [name]
```

where name is the symbolic name for a revision. rcsfreeze freezes a configuration (revision) in RCS and, optionally, assigns a symbolic name to this revision. A configuration consists of one or more RCS file revisions grouped together. This is something like a release of a software system. By grouping files together in a configuration, you can later extract these files as a group for manipulation.

Merging files and revisions. The basic syntax is:

```
rcsmerge [-kstring] [-p[rev]] [-q[rev]] [-r[rev]] [-Vn] [-xsuff] file
```

where -kstring uses keyword substitution (see the co section).
-p[rev] displays results on standard output instead of replacing working file.
-q[rev] is quiet mode.
-r[rev] merges file in revision rev.
-Vn emulates version n.
-xsuff uses suffix to characterize RCS files.
file is file to merge in.

rcsmerge places the changes between two RCS revisions into a working file specified as file. If you specify two revisions with the -r option, these are the revisions specified; if you specify one revision, it compares these files with those of the latest revision in the RCS library. Finally, if you don't specify a revision, it will merge the results of the latest two revisions into the working file.

This command is useful to see what revisions have occurred and how the two specified revisions differ. You can also use this to merge changes that are received after a revision is released and changes have been made.

Displaying log information. The basic syntax is:

```
rlog [-LRbht] [-ddate] [-l[lockers]] [-r[revs]] [-sstates]
[-w[logins]] [-Vn] [-xsuff] file ...
```

where -L ignores RCS files without locks.
-R displays only RCS filename without path.
-b displays information about the highest branch.
-h displays RCS information.
-t is the same as -h and includes descriptive text.
-ddate displays revision activities within given dates where dates are specified by ranges separated by colons.
-llockers displays information about locked revisions. If lockers

is specified as a comma-separated list of usernames, only those revisions owned by these users are displayed.
-rrevs displays information about revision specified in a comma-separated list.
-sstates displays revision information that is in state.
-wlogins displays all revisions checked in by usernames in the logins comma-separated list.
-Vn emulates RCS version n.
-xsuff uses suffixes to specify RCS files.
file ... is file or files to display information about.

rlog displays information relating to those files specified in file(s). With various switches, you can control the display of information and manipulate it accordingly. A simple and common example of using this command is:

```
$ rcslog RCS/*
```

This will display information on all files in the RCS subdirectory.

There are basic themes which run through the above commands. One of the most important is the -xsuffix option. Suffixes are merely ways to control the search order of RCS when looking for RCS files. The default is determined when RCS is installed but is usually ,v. This means that RCS will looks for files ending in ,v as RCS files. You can modify this with the use of suffixes. For example:

```
$ ci -x.rcs test.c
```

will generate and search for RCS files which end in .rcs. When you executed the above command, RCS created a file RCS/test.c.rcs, which is the RCS file created with the ci command. You can specify more than one suffix by separating them with forward slashes. For example, to create and use RCS files with suffixes .rcs and ,v, use a command like:

```
$ ci -x.rcs/,v test.c
```

This will create a file test.c.rcs. However, if you had a file named test.c,v, RCS would find this with a command like:

```
$ co -xrcs/,v test.c
```

and retrieve the file correctly. Suffixes allow you to choose your file naming conventions according to your needs.

The above commands form the base set of commands with which you can perform RCS operations. The basic user need only use ci and co to

maintain and use the RCS system. Basic functions of RCS as outlined in its the accompanying documentation are:

Store and retrieve multiple versions of text

Maintain a history of changes

Lock files and control user access

Support hierarchical versioning

Merge revisions

Provide user-friendly release and configuration control mechanisms

Minimize disk space required by storing only deltas

While most points above are self-explanatory, the last is worth brief mention. RCS, like SCCS, stores only the differences between file revisions (commonly called deltas), which minimizes the amount of disk space required while maintaining the proper relationships between file versions.

There is a manual page named rcsintro.1 which describes in some detail the operation of RCS.

There are linkages between RCS and some versions of make, particularly GNU make. See the rcs.ps file in the RCS main directory for more details on this interface.

7.12.3 Examples

A simple example of the use of RCS will explain most of the function you will need to know to use RCS effectively. Assume you have a Fortran source file named test.f. You want to check it into RCS. First create an RCS subdirectory:

```
$ mkdir RCS
```

Then check the file into RCS with the command:

```
$ ci test.f
```

This will create a file in the RCS subdirectory and remove test.f from the current directory. Remember the default first revision is 1.1. If you want to check in the file but not remove it from the current directory, use the command:

```
$ ci -l test.f
```

or

```
$ ci -u test.f
```

The first does an implicit checkout after the checkin and locks the file for editing. The second does an implicit checkout after checkin but does not lock the file.

Next, you may want to check out the file for modifications. Use the command:

```
$ co -l test.f
```

This will take the latest version of test.f from the RCS subdirectory and create a test.f file in your current working directory. Note that if you co without the -l option, you will not be able to check the file back in with a subsequent ci command. This is useful when you want to compile or examine a file without making changes.

Once you have edited the file, you can check it back in with the command:

```
$ ci test.f
```

This will remove the file test.f from your current working directory and save it as test.f revision 1.2. If you have not locked the file with co -l, you will get an error message relating to the fact that you don't own the lock on test.f. If this is the case, you can force a lock with the command:

```
$ rcs -l test.f
```

This grabs the lock without actually checking out the file. Note also that if someone else has the lock, this command will fail and you will have to see the person who has the lock and coordinate your checkin with him or her.

You can also compare a working file to an RCS file with a command like:

```
$ rcsdiff test.f
```

This will generate a diff command between your current test.f working file and the highest revision test.f in the RCS system.

There are other commands provided with rcs; however, your need for them will probably be mininimal, at least initially. Keep these simple examples in mind when beginning to use RCS.

7.12.4 Installation

There are several issues to note when installing RCS. You need a diff command that supports the -n option. If yours does not, get GNU diff from the accompanying CD and use it. There is a README file in the

main RCS directory which has an in-depth discussion of the history of features and functions of RCS. See this for related information. The information on how to build RCS is contained in the ./src/README file. A synopsis of that file follows.

There is no configure as is standard delivery with most newer GNU products. There is instead a shell script named conf.sh. This is executed from within the Makefile and builds the proper configuration as well as it can. Note that the initial section of the Makefile is called the configuration section. This is the place where you may have to change some variables and macro definitions based on your machine architecture and what pieces of software you have installed. For example, if you don't have the GNU diff command installed, you need to comment out the line DIFF and remove the comment sign from the previous line which defines standard diff. There is a section in the README file entitled Makefile notes. See this for more details on possible problems or notes for your particular platform.

The first commands to type are:

```
$ cd src
$ make conf.h CC=gcc
```

This generates a new conf.h appropriate for your machine. If you get an error message, see the README file section entitled conf.h notes. The error is contained in the src/conf.error file, which you can reference if you encounter any problems. Remember to use the CC=gcc flag with make, or you may get results that are different from those described in this section, which contains some brief comments as to possible problems or issues with the conf.h. There should be very little problem since conf.sh seems to be fairly conservative in its construction of conf.h and assumes very little about your machine.

Once you have built conf.h, issue the command:

```
$ make CC=gcc
```

This generates all the appropriate executables for RCS. Next you need to install the executables in your common directory:

```
$ make install
```

To test the installation, type:

```
$ cd ./svc
$ mkdir RCS
$ make install test
```

If this fails, type

```
$ make install debug
```

for more detailed error information.

This is all there is to installing RCS. It is really quite simple. If you find any bugs, report them to rcs-bugs@cs.purdue.edu.

7.12.5 Conclusion

RCS has many of the features of SCCS and other more attractive features and an easier interface which provides a more feature-rich and user-friendly interface. By providing a configuration management capability, RCS can assist you with your software development projects. RCS, with its ability to track and control file manipulation and modification and provide a history of all changes, will quickly become a required part of your software development processes.

7.13 CVS

7.13.1 Introduction

CVS stands for Concurrent Versions Systems and is a package which, when combined with RCS, provides version control well beyond the capabilities of standard packages such as SCCS or RCS. CVS allows a developer to version control files in multiple directories simultaneously to support large software development products.

While there are issues you must be aware of with capabilities such as parallel development support, CVS will support this kind of capability if you understand exactly what you are doing in your software development process.

7.13.2 Installation

In the README file there is some discussion of issues related to users of versions of CVS that are earlier V1.3. If you are a pre-V1.3 CVS user, you should see the README file for more details on backward compatibility issues and other related subjects. The basic changes are:

1. The CVS administration directory has changed from CVS.adm to CVS.
2. The format of the CVS/Entries file has changed and will automatically update older format files but will not generate the older formats.
3. The source files directory has changed from CVSROOT.adm to CVSROOT.

CVS recommends that you preinstall GNU diff 1.15 and RCS V5 before installing CVS. If you are a new CVS user, first you must install it with the following series of commands:

```
$ ./configure --prefix=/usr/local
```

This will generate the appropriate Makefile to build CVS. There is some discussion in the INSTALL file regarding the ndbm system. You should avoid using this system and use the emulation provided with CVS to ensure maximum portability and consistency across multiple platforms.

Next you need to build the product:

```
$ make
```

Note that this system assumes gcc, so it is not necessary to define any macros or anything else to the build procedure.

Finally, you need to install the binaries with the command:

```
$ make INSTALL= "/usr/ucb/install =c" INSTALLDATA=" /usr/ucb/install
=c -m 644" install
```

Note that this makefile assumes you are using the Berkeley version of install instead of the AIX default version of install, which is System V based. Fortunately, both are delivered with AIX. At this point you can set up a master source repository. This may be something you want your users to do; however, it may not be a bad idea to create it yourself to better control disk space allocation and other system-related issues.

First, find the directory you wish to make your repository and type the command:

```
$ ./cvsinit
```

This will prompt you for your master source repository directory and other relevant information. Note that you must have the RCS binaries in your PATH before you can execute this command.

You then need to tell your users to place the master repository root directory in their PATH if they wish to use the RCS system. The INSTALL document recommends that you actually place the CVS system itself under RCS control at this point.

To do this, move to the root of the CVS distribution directory and type:

```
$ make realclean
$ cvs import -m 'CVS 1.3 distribution/ cvs CVS CVS1_3
```

This will place CVS 1.3 under RCS control in your master repository directory. Don't forget to define CVSROOT before issuing the above commands. You can now use standard RCS commands to manipulate CVS just as you would any other system.

7.13.3 Usage

CVS provides for a much more sophisticated level of support for large software distribution efforts. With packages such as SCCS and RCS, you are forced to use a single directory for a single project. With CVS, you can have a package span multiple directories and can manage the development process accordingly.

The basic syntax for the CVS command is:

```
cvs [options] cvs_command [command options] [command args]
```

where options include:

`-H`	Displays usage information about a particular command or all commands.
`-Q`	Quiet mode.
`-q`	Somewhat quiet mode.
`-b dir`	Uses dir to locate RCS binaries.
`-d dir`	Uses dir as the CVS root directory (which is the master repository directory).
`-e editor`	Editor is invoked for log information.
`-l`	Doesn't add command to the history log.
`-n`	Doesn't change any files.
`-r`	Makes new working files read only.
`-t`	Trace mode.
`-w`	Makes new working file read-write (default).

CVS commands consist of:

`add file ...`	Adds one or more files to CVS working directory.
`admin files`	Performs RCS control functions on the source repository.
`checkout module...`	Checks out your private file copy to work on.
`commit file...`	Forces one or more file changes to the repository and updates your changes to other developers.
`diff [files]`	Displays differences between those files in the repository and those in the current working directory.
`export module`	Similar to checkout; however, doesn't include any CVS administration directories. This is most often used to generate distribution packages.
`history [files]`	Displays history of a file or files.
`import repository vendortag releasetag`	Used to import an entire distribution from an outside source.

`rdiff module`	Creates a diff (patch) file to create one version in the repository from another.
`release module`	Cancels a checkout, abandoning any changes.
`remove [files]`	Removes one or more files from the repository.
`rtag symtag`	Used to associate tags with explicit source versions in the repository. This can then be referenced by other CVS commands.
`status [files]`	Displays current file status.
`tag symtag [files]`	Used to associate tags with the closest repository versions to the current working directory.
`update files`	Updates your source code with all other changes made by other developers.

CVS command options consist of:

`-D date`	Uses the most recent version no later than date.
`-H`	Help.
`-k flag`	Modifies the default behavior of the -k flags for the RCS command. See the RCS documentation for more details.
`-l - local`	Doesn't trace through any subdirectories.
`-n`	Doesn't run any checkout/commit/tag program.
`-P`	Prunes empty directories.
`-p`	Pipes retrieved output to standard output.
`-r tag`	Uses the tag revision. tag can be either HEAD for the most current revision or BASE for the revision you checked into the current working directory.

One of the most useful commands is the CVS -H command; it prints out all help information.

CVS operates on the concept of a working directory. This means that you place yourself in a directory from which you can direct all CVS activity. If you want to work on a file, you use the checkout command and check out a file from the repository to the current working directory. From here you can modify the file any way you would like. Once you have modified the file, you can check the file back into the repository with the commit command.

CVS has several functions which provide significant additional value in the software development process. Some of the most important are:

1. Concurrent access control and tracking
2. Flexibility to support a variety of software structures
3. Symbolic naming conventions which allow you to use tags to reference software packages
4. Location independence of source files within a release
5. Support for the patch format for update distribution

Through the use of RCS concepts and commands, CVS provides an environment which supports much larger and more complex capabilities such as those described above. Without these capabilities, tools such as RCS and SCCS often fall short enough that they are not used.

There are a variety of files which are important to understand. Some of the key ones in the current working directory are:

`CVS`	Directory of administrative files
`CVS/Entries`	List and status of files in the current working directory
`CVS/Entries.Backup`	Backup file of CVS/Entries
`CVS/Repository`	Pathname to the corresponding directory in the repository
`CVS/Tag`	Contains the per-directory sticky tag or date information
`CVS/Checkin.prog`	Program to run on commit
`CVS/Update.prog`	Program to run on update

In the source repository directories, the key files are:

`CVSROOT/CVSROOT`	Global administrative files for CVS
`CVSROOT/commitinfo,v`	Programs for filtering commit requests
`CVSROOT/history`	Log of CVS transactions
`CVSROOT/modules,v`	Definitions of modules in this directory
`CVSROOT/loginfo,v`	Records programs for piping commit log entries
`CVSROOT/rcsinfo,v`	Records pathnames to templates used during commit
`CVSROOT/editinfo,v`	Records programs for editing commit log entries
`Attic`	Directory of removed source files
`#cvs.lock`	Lock directory
`#cvs.tfl.pid`	Temporary lock file for repository
`#cvs.rfl.pid`	Read lock
`#cvs.wfl.pid`	Write lock

Finally, other than placing the CVS commands in your PATH, there are several environmental variables which you can use to control CVS's behavior. They are:

`CVSROOT`	Defines the source repository.
`CVSREAD`	Checkout and update will make files in your current working directory read only.
`RCSBIN`	Pathname to RCS commands.
`EDITOR`	Specifies the editor to use for logging purposes.

There are many other aspects to CVS that you need to understand to use it effectively in a project environment. There are several man pages in the man subdirectory, including cvs.1. This outlines in great detail all options and gives several examples of how to use the CVS system in a project. Also there is a paper in the doc subdirectory called cvs.ps. This is a Postscript file which outlines an actual development project

that used CVS and what they discovered. This is a very useful file if you are thinking of using CVS for a project.

7.13.4 Conclusion

CVS is a very powerful tool and adds significantly to the power of RCS. By supporting multiple directories in a single release, CVS provides support for large and complex software development projects. This is a tool you may want to seriously investigate if you are interested in software development and revision control.

7.14 Smalltalk

7.14.1 Introduction

Much has been made of Smalltalk in the last few years as commercial businesses have begun to make the move to object-oriented development. Smalltalk was one of the first object-oriented languages available, and it still has a dedicated following in the object-oriented world. Because of this, this section discusses GNU Smalltalk.

GNU Smalltalk attempts to conform to the Smalltalk 80 implementation specifications as described in *Smalltalk-80, the Language and its Implementation* by Adele Goldberg and David Robson. This is the bible of Smalltalk syntax and structure and is, therefore, the one which the GNU Smalltalk attempts to support.

7.14.2 Usage

The best documentation on the usage of GNU Smalltalk is in the texinfo files. This is a texinfo document which outlines in great detail examples and issues both in terms of generic Smalltalk and the GNU implementation. This is the first document to print and read. Once you have done this, you will be much more capable of using the GNU Smalltalk tool than before. If you need information, see Sec. 8.3 for more details.

The GNU Smalltalk system is an interpreter. This means that you can interactively enter Smalltalk commands and display the results as you proceed. The first step to using GNU Smalltalk is entering the interpreter. Use the command:

```
$ mst -q
Smalltalk 1.1.1 Ready
st>
```

You are now sitting at the Smalltalk prompt, and it is waiting for input. Note that the -q option tells the interpreter to suppress much of its in-

formational output. If you want to know various aspects of the interpreter's operational context simply use the mst command without the -q option.

Another command you must first type before proceeding with the examples shown below is:

```
st> Smalltalk at: #x put: 0 !
```

More on this later.

A simple example as documented in the *.ps document is to make Smalltalk display the Hello, World string to you. Use the command:

```
st> 'Hello, World' printNl !
Hello, World
st>
```

Examining the above command tells you a lot about how Smalltalk operates. The Smalltalk interpreter processes all text that precedes an exclamation point as a single command. In this case, the command is 'Hello, World' printNl. The interpreter actually creates a default object of type string which contains the string Hello, World. It then passes the message printNl to the default object which then executes the message as a procedure inside the object string. This results in the Hello, World string being displayed on your screen.

What this means is that the default object string has a listing of possible messages it can be passed. When the object received a message, it compared it to the list of possible messages and executed code based on the message it received.

As discussed in mst.texinfo, you can pass virtually any type of object to the interpreter and expect that it will display itself in a similar manner. For example, you might use the integer object like:

```
st> 10 printNl !
10
st>
```

This displays exactly the information passed to the object of type integer.

Smalltalk supports arrays much as languages such as C do; however, the syntax is dramatically different. To create an array, use a command like:

```
st> x := Array new: 10 !
```

This will create a 10 element array. Next, to view the contents of a member of the array, use a command like:

```
st> (x at: 1) printNl !
nil
st>
```

This shows that array elements are initialized to nil. Next you may want to enter some element data. Use a command like:

```
st> x at: 1 put: 10 !
```

This will store 10 in the first element of the array. As shown above, many messages can be passed arguments which consist of syntax like:

```
message: argument
```

where the message is followed by a colon and then an argument. This will be fairly consistent throughout your usage of the Smalltalk interpreter.

Note that array elements can be used exactly as you would use any other variable. For example:

```
st> ((x at: 1)+ 1) printNl !
11
```

This takes the contents of the first array element (in this case 10), adds 1 and displays the resulting value of 11.

The other main construct you can use in Smalltalk is the set construct. After you have finished playing with the x array as described above, you can reassign the x variable to a set. The array previously bound to x will be automatically destroyed by the Smalltalk interpreter in a process known as garbage collection. This is similar to the garbage collection used in other object-oriented languages such as C++ and Eiffel and will work approximately the same way.

To create an empty set bound to the variable x, use:

```
st> x := Set new !
```

This will create an empty set . To view the empty set, use the command:

```
st> x printNl !
Set()
st>
```

This denotes that the set is empty. To add information to the set, use a command like:

```
st> x add: 5; add: 7;add: 'foo' !
```

```
st>
```

Now display the contents of the set:

```
st> x printNl !
Set (5 'foo' 7)
```

Note that this represents only the unique elements in the set. To remove an element from the set, use the remove message as shown below:

```
st> x remove: 5 !
st> x printNL !
Set ('foo' 7)
```

The final construct this section will talk about is the dictionary. Everything in Smalltalk is driven from a stored dictionary of information and relationships. A dictionary can be indexed by anything, while an array can only be indexed by integer. For example:

```
st> x := Dictionary new.
st> x at: 'One' put: 1 !
st> x at: 'Two' put: 2 !
st> x at: 1 put: 'One' !
st> x at: 2 put: 'Two' !
```

To examine the dictionary, use the printNl command again:

```
st> x printNl !
Dictionary(1,'One' 2,'Tow' 'One',1 'Two',2)
```

This displays the dictionary as sets of keys and associated values. Note that you can reference the values by simply using the key as you would the integer of an array. For example:

```
st> (x at: 1) printNl !
'One'
st>
```

The first command you typed consisted of a definition for a Smalltalk dictionary. Smalltalk is a special variable which both the user and interpreter can access and manipulate. The #x variable represents a true variable that can be substituted for at any time. Unlike specifying a simple x, the #x represents something that can change over time. For example, you can create a new definition with a command like:

```
st> Smalltalk at: #y put: 0 !
```

You can now manipulate y just as you have been doing with x.

To display the entire Smalltalk dictionary, you can use the command:

```
st> Smalltalk printNl !
```

You can CTRL-C at any time to get back to the st> prompt.

This section has merely paraphrased Sec. 2 of Andrew Valencia's paper included in the Smalltalk distribution. You should reference this document for more information on Smalltalk and its capabilities.

7.14.3 Installation

The source distribution for GNU Smalltalk consists of several directories, including config, contrib, examples, test, and stix. These directories include supported platform-specific files, contributed software, example Smalltalk files, regression testing files, and an XII interface to Smalltalk, respectively. The main source code for the product is in the top directory of the distribution and consists of both Smalltalk (.st) and C source code (.c) files.

Based on some of my surfing on the Internet in the comp.lang.smalltalk and gnu.smalltalk.gnu newsgroups, the support for GNU Smalltalk is not high, and you, therefore, may have to do some experimenting on your own to get things compiled and working correctly. However, this does not minimize the power of this tool.

7.14.4 Conclusion

While this has not been an exhaustive account of GNU Smalltalk, it has introduced you to the concepts and capabilities of the GNU Smalltalk implementation. Acquire and install the product for more information. See also the *.ps file written by Andrew Valencia included with the Smalltalk distribution. It is an excellent overview of Smalltalk and the GNU Smalltalk implementation.

The paradigm of object-oriented technology is radically different from that of the procedural paradigm we have been following since the 1970s. Smalltalk is a tool which can assist you in the move toward object-oriented development.

7.15 f2c

7.15.1 Introduction

f2c is a program which attempts to convert Fortran to C. It is not clear how good this software is; however, it has been recently updated on the network, so there is work going on at BellCore and AT&T. There is a disclaimer issued with the product which is contained both in the prod-

uct and in App. C in this book. While I have not personally used this product, it does look interesting and, if it works, could be quite useful. The best way to check the usefulness of this system is to try it and see what kind of code it produces.

f2c is an AT&T/Bellcore product and as such is subject to its AT&T/Bellcore copyright as included both in the product distribution and in App. C of this book.

7.15.2 Usage

Much of the information in this section is taken from the file f2c.1, which is a formatted manual page. You can print this out or view it on line for more information.

The basic usage of f2c is simple:

```
f2c [-C] [-I2] [-onetrip] [-C++] [-ec] [-ext] [-i2] [-r8] [-Tdir]
[-w8] [-Wn] [-!c] [-!I] [-!it] [-!P] [-AEPRUacgpuwz] file ...
```

where -!P doesn't infer ANSI or C++ prototypes from usage.
-!c doesn't produce C code.
-!I doesn't include include statements.
-!it doesn't infer type of untyped EXTERNALs based on their use as parameters to previously defined or prototyped procedures.
-A produces ANSI C (default is K&R C).
-C ensures subscripts are within array boundaries.
-E declares uninitialized COMMON to be extern.
-I2 defines INTEGER and LOGICAL as 2 bytes.
-onetrip compiles DO loops that are executed at least once.
-P creates file.P contining prototypes.
-R doesn't make REAL become DOUBLE PRECISION.
-Tdir places temp files in dir.
-U maintains case of variables and external names.
-Wn defines n characters per word.
-a makes local variables automatic.
-c includes original Fortran source code as comments.
-C++ generates C++ code instead of C.
-ec—uninitializedCOMMON placed in separate files.
-ext issues messages concerning non-F77 Fortran code.
-g includes original Fortran line numbers as comments.
-i2 is similar to -I2, only INTEGER and LOGICAL may be assigned by INQUIRE.
-p includes preprocessor definitions too make COMMON block members look like local variables.
-r8 promotes REAL to DOUBLE and COMPLEX to DOUBLE COMPLEX.

-u makes the default type undefined.
-w suppresses all warning messages.
-w8 supresses warnings when COMMON or EQUIVALENCE forces odd word alignment of doubles.
-z doesn't implicitly recognize DOUBLE COMPLEX.
file... is one or more files to be processed.

The input files will be scanned for .f and .F files with a filename matching file. An output file will be created with the name file.c. You should be able to compile this resultant file.c with your C compiler and proceed as you normally would.

Note that the resulting C routines require both the libI77 and libF77 archive libraries when they are linked. This means that you need to use the -lF77 -lI77 -lm options on the ld or cc command to ensure that these programs are properly linked.

A simple example consists of a file named test.f, which is a Fortran file, as shown below:

```
      PROGRAM MAIN
C
C     THIS IS PROGRAM TEST
C     IT CALCULATES THE SUM OF UP TO N VALUES OF X**3 WHERE
C     NEGATIVE VALUES ARE IGNORED
C
      READ(5,*) N
      SUM=0
      DO 10 I=1,N
        READ(5,*) X
        IF (X.GE.0) THEN
           Y=X**3
           IF (SUM.GE.0) THEN
             SUM=SUM+Y
           ELSE
             GOTO 20
           END IF
        END IF
10    CONTINUE
20    CONTINUE
      WRITE(6,*) 'This is the sum:',SUM
      STOP
      END
```

You invoke the f2c converter program on it with the command:

```
$ f2c test.f
```

This creates a file test.c which looks like:

```
/* test.f -- translated by f2c (version of 13 June 1995 18:34:30).
 You must link the resulting object file with the libraries:
   -lf2c -lm (in that order)
*/
#include "f2c.h"
/* Table of constant values */
```

```
static integer c__3 = 3;
static integer c__1 = 1;
static integer c__4 = 4;
static integer c__9 = 9;
/* Main program */ MAIN__()
{
 /* System generated locals */
 integer i__1;
 real r__1, r__2;
 /* Builtin functions */
 integer s_rsle(), do_lio(), e_rsle(), s_wsle(), e_wsle();
 /* Subroutine */ int s_stop();
 /* Local variables */
 static integer i, n;
 static real x, y, sum;
 /* Fortran I/O blocks */
 static cilist io___1 = { 0, 5, 0, 0, 0 };
 static cilist io___5 = { 0, 5, 0, 0, 0 };
 static cilist io___8 = { 0, 6, 0, 0, 0 };
/* THIS IS PROGRAM TEST */
/* IT CALCULATES THE SUM OF UP TO N VALUES OF X**3 WHERE */
/* NEGATIVE VALUES ARE IGNORED */
 s_rsle(&io___1);
 do_lio(&c__3, &c__1, (char *)&n, (ftnlen)sizeof(integer));
 e_rsle();
 sum = (float)0.;
 i__1 = n;
   for (i = 1; i <= i__1; ++i) {
    s_rsle(&io___5);
    do_lio(&c__4, &c__1, (char *)&x, (ftnlen)sizeof(real));
    e_rsle();
    if (x >= (float)0.) {
/* Computing 3rd power */
    r__1 = x, r__2 = r__1;
    y = r__2 * (r__1 * r__1);
    if (sum >= (float)0.) {
      sum += y;
    } else {
      goto L20;
    }
   }
/* L10: */
 }
L20:
 s_wsle(&io___8);
 do_lio(&c__9, &c__1, "This is the sum:", 16L);
 do_lio(&c__4, &c__1, (char *)&sum, (ftnlen)sizeof(real));
 e_wsle();
 s_stop("", 0L);
} /* MAIN__ */
/* Main program alias */ int main_ () { MAIN__ (); }
```

Once you have created the test.c file, you need to compile and link the resulting executable with a command like:

```
$ gcc -o test test.c -lF77 -lI77 -lm
```

Note that the F77 and I77 library archives must either be in the default /usr/lib, if not, you must include an -L command like:

```
$ gcc -o test test.c -L/usr/local/f2c/libF77 -lF77
-L/usr/local/f2c/libI77 -lI77 -lm
```

Note that the -L directories must be wherever you have built the archive libraries included with this distribution.

The best set of examples and discussion of feature/function for f2c is in the document f2c.ps, which is a Postscript document included with this distribution. If you have Ghostscript installed, you can preview it. You can also print it to any Postscript printer for review.

If you have problems with unresolved references, it may be because of the way your compiler generates labels, and the necessary conversion to link C and Fortran routines together into a single executable. See your compiler documentation for more details. It is important to use the same compiler to generate both the libraries and the final executable. gcc is a good compiler and worked fine on Sun and IBM machines. Other machines were not tried but are supported. Keep this in mind as you attempt to build final executables.

The above example file is included on the accompanying CD in the ./src/example directory. This is something I added in before distribution and in no way is related to the original distribution of the f2c system.

There is a file named README which you should examine since it has some information related to the function of f2c, including a discussion of bug reports, machine dependencies, and system call issues. See this file before building the product. See also a file named fixes. This contains information on fixes and functionality in various versions of the product.

7.15.3 Installation

This product was built on several UNIX machines, including an RS/6000. As with most products, there are some subtleties to watch out for. The first recommendation is to use the GNU C compiler since it will ensure that ANSI file headers and other constructs will be supported by your compiler. You can try to build the products with your native compiler, but if you have trouble, try it with the GNU C compiler.

First move to the ./src subdirectory and build the xsum and f2c executables with a commands:

```
$ make clean
$ make all
```

If you want to use the GNU C compiler, you must invoke commands like:

```
$ make clean
$ make all CC=gcc
```

You cannot change the makefile contained in the ./src subdirectory because the makefile itself checks the contents of the makefile to ensure that it was transmitted over the network correctly. While this is unfortunate, you can modify the C compiler choice by using the macro command line subsitution with make as described above.

Next you should move to the ./libF77 subdirectory to compile this library. As is documented in the ./libF77 README file, the makefile assumes that f2c.h header file is contained in /usr/include. As before, ensure that there are no files in the directories which can change the results of the makefile with the command:

```
$ make clean
```

The next issue is to create the proper f2c.h file in the ./libF77 subdirectory. Unfortunately, the makefile hard-codes the location of f2c.h as /usr/include. If you do not want to place f2c.h in this directory, you must first issue the following command:

```
$ cat /usr/local/f2c/f2c-1993.04.28/src f2ch.add > f2c.h
```

This assumes that your f2c is installed in /usr/local. If you have placed the distribution in some directory other than this, you must substitute your path for /usr/local in the above command. Note also that you need to be sitting in the ./libF77 subdirectory when you issue the cat command. This command creates a new f2c.h in the local directory which you will use to create the libF77.a archive library. Use the make command to build the archive library:

```
$ make CC=gcc
```

Remember that you can use your default C compiler by simply issuing the make command without the CC macro definition.

On an RS/6000 there is a bug with the assembler unless you have PTF U416277 installed. If you do not have this installed, you need to override the -g option on the compile statement with a command like:

```
$ make CC=gcc CFLAGS=
```

This will override the default in the makefile and build the f2c system without debug information. In fact, this is a good idea anyway since it creates a more efficient system without the -g option. Note that you must do the same thing for the other subdirectories (libF77 and libI77).

Finally, move to the ./libI77 subdirectory and issue the same commands you issued to build libF77. There is one change. You must create an empty file named local.h in the ./libI77 subdirectory since the fp.h

include file attempts to include it. This file is useful only for VAX and CRAY environments, and you therefore only need to create an empty file with a command like:

```
$touch local.h
```

Now use the make command to create the archive library. Once you have done this you have completely constructed the f2c distribution.

See the corresponding README files for each subdirectory for more information related to their contents.

7.15.4 Conclusion

f2c is a tool which attempts to take Fortran code and automatically convert it to C code. The basic syntax seems to be supported fairly well; however, there is simply not enough experience or information available as of the time of this writing to confirm its stability. Simply try it for yourself on some of your codes and see what happens.

Whatever happens, it may be a very useful tool for you to use in beginning the port of some of your Fortran codes to the C language and environment. See the file f2c.ps in the distribution for more information.

7.16 Ftncheck

7.16.1 Introduction

Ftncheck is a freely available tool which significantly enhances the productivity of a typical Fortran programmer. Unlike many tools, including most Fortran compilers, Ftncheck doesn't check the syntax of the Fortran programs it analyzes. Instead, it provides semantic checking which enables developers to catch logic errors and potential run-time problems well before execution of the application. Through the use of Fortran logic and other analysis techniques, Ftncheck can alert the developer to conditions like unused or uninitialized variables, strange loop constructs, and other wasteful or incorrect programming procedures.

7.16.2 Usage

The basic syntax for Ftncheck is:

```
ftncheck [-[no]declare] [-[no]division] [-[no]extern] [-[no]f77]
[-[no]library] [-[no]linebreak] [-[no]list] [-[no]portability]
[-[no]project] [-[no]sixchar] [-[no]symtab] [-[no]usage]
[-[no]verbose] [-output=file] [-columns=num] [-common=num]
[novice=num] file...
```

where -declare displays a list of all identifiers whose datatype is not explicitly declared.
-division warns wherever division is done.
-extern warns if external subprograms are never defined.
-f77 warns about violations of the F77 standard.
-library begins library mode. Does not warn if subprograms are defined but never used.
-linebreak treats line breaks in continue statements as spaces.
-list displays source listing.
-portability warns about nonportable usages.
-project creates project file.
-sixchar displays variable names which clash at six characters in length.
-symtab displays symbol table.
-usage warns if variables are not used.
-verbose is verbose mode.
-output=file places output in a file.
-columns=num sets maximum line length to num.
-common=num—strictness in checking COMMON blocks. Minimum is 0; maximum is 3.
-novice=num sets novice level. Minimum is 1; maximum is 5.
file... specifies one or more files to analyze. Be careful concerning extensions (see below).

There are a tremendous amount of defaults included in the logic of Ftncheck, including automatic input filename extensions (.prj, then .f in UNIX) which significantly enhance the usability of Ftncheck. The .prj files are project files. These are created by Ftncheck and are very useful in storing information which is useful in building multifile analysis jobs. There is a more thorough discussion of the options, including project files, in the included manual page and Postscript document. See these for more details on the individual options.

As with most applications, the best way to learn Ftncheck is through examples and actual use. A simple example of Ftncheck is:

```
$ ftncheck -library -noextern -project test.f
```

This will read the input file test.f and create a project file which contains a variety of information regarding test.f itself. The file created will be named test.prj. This is called a project file and can be used in the incremental process of analyzing a large or changing group of files for consistency across the source code files. This is one of the most useful functions of Ftncheck and should be utilized to the fullest, especially when building large software systems.

If you used the command:

```
$ ftncheck -library -noextern test test1.f
```

Ftncheck would first take the results from the project file test.prj and cross-reference them with the output of the test1.f analysis. This allows you to save significant time when analyzing changing systems.

You can also modify the default behavior of Ftncheck through the use of variables. To modify the behavior of a variable, simply precede the option name with FTNCHECK_ and ensure that you export the variable (or use setenv in the C shell) to ensure that it is passed to subsequent shells. For example, to see the columns to 132, use the commands:

```
$ FTNCHECK_COLUMNS=132
$ export FTNCHECK_COLUMNS
```

or

```
$ setenv FTNCHECK_COLUMNS 132
```

depending on the shell you are using. Note that you have to use the complete option name in uppercase for this methodology to work.

There is an excellent example of the output of Ftncheck in the manual page. Duplicating this would be a waste of paper, and therefore, I strongly urge you to examine the accompanying man page for a good example of Ftncheck output.

Earlier in this section it was stated that Ftncheck does not perform syntax checking. This is not totally true. Ftncheck does perform some basic syntax checking along with other more logic-oriented checks. Ftncheck has four main types of messages:

1. Portability warnings
2. Other warnings
3. Informational messages
4. Syntax errors

Portability warnings outline potential portability problems with the code regarding the particular use of nonstandard logic. Other warning messages display messages not normally displayed by the compiler. Informational messages consist of warnings which will directly assist developers in debugging their code. Finally, syntax errors may be detected and presented to the user; however, Ftncheck does not catch

all syntax errors and should be used only after the compiler has correctly compiled the program.

Along with a much more thorough discussion of Ftncheck's functions and capabilities, the man page provides information on current release functionality as well as any known bugs. There are a variety of bugs in the system, and you should check the end of the man page before using Ftncheck to save yourself some major headaches.

7.16.3 Installation

The installation of Ftncheck is very straightforward. Edit the makefile and modify the BINDIR and MANDIR to your appropriate directories. Then issue the command:

```
$ make IBM-RS6000
$ make install
```

This will generate the Ftncheck binary. Note that Ftncheck is written in C and obviously requires a C compiler to build properly. Either the native AIX C compiler or the GNU C compiler system will work well.

7.16.4 Conclusion

Ftncheck is an extremely useful tool in terms of checking the logic and flow of Fortran programs. By providing fairly extensive semantic checking of Fortran programs, Ftncheck can signficantly aid in the run-time debugging and tuning of Fortran software systems. This is definitely a tool which will be of great value to almost any Fortran programmer.

7.17 imake and xmkmf

7.17.1 Introduction

imake is a facility which generates makefiles in conjunction with cpp (the C preprocessor). This allows you to generate different makefiles based on cpp directives such as #define and #include. This provides you with the ability to include and define the appropriate information in your makefiles so they will build correctly. When you use a heterogeneous environment, this becomes a very powerful tool to build portable makefiles where you can have imake determine what type of machine you are on and build the appropriate makefile. This powerful feature is often used by the more complex and sophisticated packages on the Internet.

imake is used by older software systems from the Internet and became fairly widely used because the X11 distributions used it to generate their makefiles. In fact, imake still comes with distributions of X11.

The distribution included with this book is a subset of those commands directly related to the imake and associated commands.

imake is part of the X11 distribution and as such is copyrighted by M.I.T. (Massachusetts Institute of Technology). They have provided this technology free, given the M.I.T. copyright notice in App. C. See this or the software distribution itself for more details.

7.17.2 Usage

The basic syntax for imake is:

```
imake [-Ddefine] [-Idir] [-Ttemplate] [-f file] [-s file] [-e] [-v]
```

where -Ddefine allows you to issue #define definitions to pass to cpp.
-Idir—dir is the directory of the Imakefile templates.
-Ttemplate is the template is used to give the name of the template files.
-f file—file is the per directory input template (default Imakefile).
-s file—file is the name of the output file (default is Makefile).
-e executes the Makefile after it is generated (this is not the default).
-v is verbose mode.

imake uses a variety of files to generate the final makefile. It works by placing a preprocessor file named Imakefile in each directory which contains something to be built. imake also uses a generic template file named Imake.tmpl to get generic machine-specific information. Finally, Imake.rules contains information on a specific platform which will give imake the ability to generate exactly the makefile necessary to build on a particular platform. All files are passed first through cpp to preprocess any cpp directives. The results of this preprocessing pass are incorporated into the resulting file Makefile.

When you invoke imake, it passes all variables contained in the -I and -D command line options as well as the following lines:

```
#define IMAKE_TEMPLATE "Imake.tmpl"
#define INCLUDE_IMAKEFILE "Imakefile"
#include IMAKE_TEMPLATE
```

The convention for variable definitions is that imake variables are mixed case, while make variables are uppercase.

The Imake.rules file contains cpp preprocessor macros of the form:

```
string(s) @@\
```

An example, as outlined in the imake man page, might be something like:

```
#define     program_target(program, objlist)   @@\
program:    objlist                            @@\
            $(CC) -o $@ objlist $(LDFLAGS)
```

When you call it with a command like program_target(foo, foo1.o, foo2.o), it will be expanded to:

```
foo:    foo1.o foo2.o
        $(CC) -o $@ foo1.o foo2.o $(LDFLAGS)
```

You can see that the rules file is important to allow you to define your own cpp macros in your Imakefiles. For a good example of a complex Imake.rules, see the ./config subdirectory on the accompanying CD.

There are several environmental variables which can be used to alter the default behavior of imake. Some of the most important are:

IMAKEINCLUDE	Any valid directory for include files to cpp
IMAKECPP	cpp to use, including fully qualified pathname
IMAKEMAKE	make to use, including fully qualified pathname

You can use macro definitions and redirections as you normally would with any cpp file and directives.

For more information on imake and its associated files, see the manual page for imake in the config subdirectory included with the X11R5 imake distribution. This directory also contains a vareity of cf (configuration) files for different architectures which help to define variables for several different kinds of UNIX machines.

The misc subdirectory contains a variety of files which contain notes for porting imake and its associated files to a variety of different platforms. While this provides much of the capabilities for a limited number of machine, you will probably need to some of your own definitions and learning as you move imake to your particular platform.

There are a variety of other tools in the scripts subdirectory, including the xmkmf command. The basic syntax of this command is:

```
xmkmf [-a] [topdir [curdir]]
```

where -a builds the makefile in the current directory and invokes any subdirectory makefiles automatically.
topdir is the root directory where makefile search should begin.
curdir—you can set this to allows xmkmf to reference any directory as its current directory.

xmkxf is an easy command interface to imake. It makes assumptions as to its directory structure and the location of imake template files. This is often run by X11 distributions to generate the proper makefiles

from the Imakefiles. A simple example of how xmkmf and imake are related is as follows:

```
$ ../../config/imake -I../../config -DTOPDIR=../../. -DCURDIR=./lib/X
```

or

```
$ xmkmf ../../. ./lib/X
```

These two commands accomplish the same thing. This is taken directly from the man page for xmkmf and is very demonstrative of the simplicity of xmkmf. Most machines that run X Windows come with this command, although they may not have a man page. Look in the X11 area for the file xmkmf and the subsequent man page for more information.

Another command that is often useful is the imdent command. This is a script contained in the scripts subdirectory. imdent lists all cpp directives and their associated nesting level. This allows you to see how your Imakefiles are structured and to ensure that you don't have any problems due to nesting problems. The basic syntax is:

```
imdent [-n] [file...]
```

where -n specifies the number of spaces used to designate indentation between nesting levels. Default is two spaces.
file... is one or more files to analyze. Default is standard input.

The final command of interest with this distribution is the mkdirhier command. This will create any directory hierarchy, including any intermediate subdirectories as necessary. This is different from the mkdir command, which forces you to create all intermediate subdirectories. The basic syntax is:

```
mkdirhier file
```

where file is any directory you wish to create.

This command is very useful when creating nested subdirectories. Unlike the mkdir command, it is not necessary to create any intervening subdirectories before creating the file subdirectory. For example:

```
$ ls -R
file1 file2
$ mkdirhier ./sub1/sub2
$ ls -R
fil1 file2
./sub1:
sub2
```

Note that, unlike mkdir, it was unnecessary to create the sub1 directory and then create the sub2 directory. This is very useful when installing software and building complex systems.

7.17.3 Installation

Both imake and xmkmf are included with AIX but in uncompiled form. To build imake, you need to type:

```
$ cd /usr/lpp/XII/Xamples/config
$ make -f makefile.ini
```

This generates imake in this directory for you. You can move the resulting files anywhere in your filesystems.

To generate xmkmf, use the following commands:

```
$ cd /usr/lpp/XII/Xamples/util/scripts
$ ../../config/imake -I../../config -DTOPDIR=../../
```

This generates a makefile from the intake templates in the scripts directory. Now you can use the following commands to build xmkmf.

```
$ make
```

This is all there is to do to generate both imake and xmkmf on AIX.

7.17.4 Conclusion

imake and xmkmf are very powerful facilities which are typically shipped with X11 distributions because of their dependency on them. AIX includes in source code format by default. Use them after you build them as described earlier.

imake allows you to use cpp directives within a makefile and preprocess the appropriate information into the resulting makefile for a specific platform. It is a very powerful tool and is something to consider when distributing software systems to heterogeneous environments.

Chapter

8

General Tools

While other chapters in this book discuss topics directly relevant to a particular development area or type, this one discusses tools which don't really fit into any specific category. Tools such as spreadsheets, documentation tools, interactive shells, and input/output tools are documented here. There are many more good tools which provide function beyond those offered here. This is merely an introduction to the types of tools that are available on the Internet.

8.1 oleo

8.1.1 Introduction

oleo is a very sophisticated and powerful spreadsheet package provided by GNU for most UNIX flavors. oleo works much as other spreadsheets work with cells referenced by characters and numbers and using macros to perform common functions. oleo 1.5 has a significant number of bug fixes from the previous releases and seems to be much more robust and stable than the previous ones.

oleo supports both dumb terminals through a curses interface and X11. You can generate embedded Postscript files which are snapshots of all or a portion of the spreadsheet. They can then be printed, displayed, or faxed with other tools such as Ghostscript and Netfax.

oleo is a GNU product and as such is subject to its GNU General Public License as included both in the product distribution and in App. C of this book. The relatively new version of perl is 5.001; older versions including 4.006 are still available on the net if you need some of that function.

8.1.2 Usage

oleo looks very much like other spreadsheets on the surface but has much more capability than other similar products. The basic input screen is shown in Fig. 8.1. There is a listing of main commands in the USING file. See this for a detailed breakdown of basic commands.

The other file which contains useful information is the KEYS file. This contains a description of all keystrokes and their associated commands. The syntax used below is the same as that used in the emacs chapter (Chap. 9). Namely, the C-n represents the CTRL character held down while the n key is pressed. This is the same nomenclature as that used by emacs.

The basic oleo commands are:

`C-g`	Aborts the current command
`C-X C-C`	Ends the session
`C-Z`	Suspends oleo
`C-x !`	Recalculates
`C-H`	Help mode
`C-h C`	Help with a command
`C-X j`	Moves the cursor to a specific address
`C-P`	Moves cell cursor up
`C-N`	Moves cell cursor down
`C-F`	Moves cell cursor right
`C-B`	Moves cell cursor left
`C-[ p`	Scans up

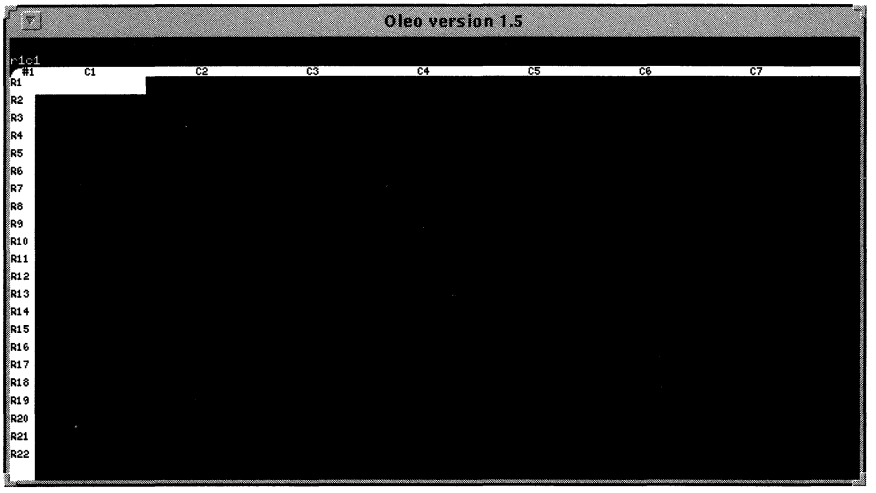

Figure 8.1 The oleo screen.

C-[n	Scans down
C-[f	Scans right
C-[b	Scans left
C-[v	Scrolls up
C-V	Scrolls down
C-X >	Scrolls right
C-X <	Scrolls left
C-X (	Begins entering a macro
C-X)	Ends entering a macro
C-h o	Displays options
C-[C-P a	Saves region as ASCII
C-[C-P p p	Saves region as Postscript
C-[C-P p s	Saves page size for Postscript printer
C-[g	Interface to gnuplot, if installed, so you can plot results
C-[c	Copies region
C-[m	Moves region
C-O	Inserts row
C-[o	Inserts column
C-K	Deletes row
C-[k	Deletes column
M-x clear-spreadsheet	Clears the spreadsheet
C-X 5	Splits window horizontally
C-X 2	Splits window vertically

There are a variety of other commands which provide more sophiticated function from oleo. See the KEYS file for a complete listing.

There is a file named USING which contains a description of basic oleo operations and functionality. Reference it for more detailed information on oleo. This section will provide a very brief overview merely to introduce you to basic oleo capabilities. The basic oleo command structure is similar to emacs and uses both the escape and control keys.

oleo operates on a cell basis much like Lotus or Excel, and the command syntax is virtually identical. Keyboard mapping can be controlled and changed based on user need. You toggle between input and command mode.

oleo supports a variety of input and output formats including integer, float, hidden, graph, general, dollar, comma, percent, and several others. Because of the power of the underlying UNIX architecture, oleo has a variety of functions beyond what most spreadsheets provide. For example, you can have the spreadsheet automatically calculated while continuing to enter input. You can also have automatic backup generation and automatic update of various functions such as rnd(), cell(), my(), and others.

8.1.3 Installation

oleo uses the standard configure package that comes with most GNU utilities. As with other configure utilities, you should use a series of commands like:

```
$ cd oleo/oleo-1.5
$ configure
$ make clean
$ make
```

This will build oleo in the current subdirectory. To install into your default directory, use a command like:

```
$ make install prefix=/usr/local
```

This will copy the file oleo to the /usr/local/bin subdirectory.

There are a few special instructions for the build of oleo documented in a file INSTALL.OLEO. See this for detailed information. The main options for compilation include spreadsheet size, X11 support, and a few machine-specific instructions regarding include files. However, with AIX, the build is very simple and seamless, and the commands as listed above will work well to build and install the product.

One thing to note is a bug which exists on AIX. You need to have the same PTF that needs to be insalled for gcc and some other programs (PTF U416277) from IBM. If you don't have this on your system, you will need to either remove the -g option from the CFLAGS macro in the Makefile or issue the make like:

```
$ make CFLAGS=
```

This will fix the related problem on the RS/6000.

This installation is fairly straightforward and presents no real problems. If it does, see the GNU people for more information. Also see the README file for possible contacts and mail addresses.

8.1.4 Conclusion

oleo is simply one of the most powerful spreadsheet programs available today. While it supports syntax used by tools like Lotus and Excel, it provides additional functionality that you may want in a spreadsheet. Plus it's *free.*

This has certainly not been an exhaustive presentation of oleo and its capabilities. However, it has presented some of its basic capabilities and characteristics. Keep oleo in mind when you are evaluating spreed-

sheets. By using oleo, you can significantly reduce your need to purchase spreadsheet tools and facilities.

8.2 perl

8.2.1 Introduction

perl stands for practical extraction and report language (but is better known as pathologically eclectic rubbish lister). perl really consists of the best of many structures and constructs from C, sed, and awk as well as many of the functions of shell programming. It is a language unlike any other you have seen and, subsequently, has a significant amount of power and capability for doing work on a UNIX platform.

perl is a massive and powerful utility that is fairly new in the UNIX community. Because of its power, this section will not spend very much time on its use and syntax but will focus more on its overall capabilities and function. There is a 100+ page manual page included with the perl distribution, and this covers all perl capabilities and syntax in great detail.

perl really provides functionality that combines the best of C programming with the best of shell programming. It provides a rich data manipulation capability as well as data display capability. As perl matured, it gained the capability to use sockets for network and interprocess communication as well as file and process manipulation.

perl makes easy what has been tradionally difficult to do in the past: combine interactive applications to manipulate and display information not only on the local machine but on remote machines as well. perl provides a conglomeration of many UNIX functions in one C-like language. If you are familiar with C syntax, perl will look very familiar and should be relatively easy to learn. However, because of the complexity, it will take some time to learn all of the functions available with perl, so be ready to dedicate a few days to become proficient.

perl supports many of the structures and functions of sed and awk and in fact contains tools to covert your old awk and sed scripts to perl scripts. This is a very useful first step in learning perl since it provides you with a direct translation from something you are familiar with to something you are trying to learn. From this you can see how perl functions and the advantages it can provide.

perl also delivers a debugging system which allows you to trace your programs in real time and see its behavior one line at a time.

The benefits of perl in a heterogeneous computing environment are clear: one consistent development interface across all platforms. There are many differences between point tools in a UNIX solution. Some machines use ksh while others use sh. Some machines use awk while oth-

ers use nawk. Because of UNIX's ability to pipe tools together, developers tend to string single tools together in a long chain to accomplish their desired result. As the heterogeneity of the environment increases, this chain begins to break down as different point tools behave differently on different platforms. perl fixes this by providing a single, consistent interface across all platforms, including most of the point tools within its domain. This is probably the biggest single benefit of perl for most power users and is the main reason perl is so widely accepted.

perl is a GNU product and as such is subject to its GNU General Public License as included both in the product distribution and in App. C of this book.

8.2.2 Installation

perl comes in a compressed tar file. First unzip the file and unwind the resulting tar file. Once this is done, run the configure command:

```
$ ./Configure
```

This is not the typical configure system which generates makefiles and determines the type of machine you are running on by itself. This is a script which prompts you for input at a variety of places and lets you help it figure out what kind of machine it is on.

There are only a couple of issues to be concerned with in the Configure script. The first is that you should not select aix for any questions since this will not work. Instead of aix, select none. For example:

```
Which of these apply, if any [aix] none
Operating system name [aix] none
```

You need to select none for both of these questions to ensure that perl will build correctly.

Finally, do not select dynamic loading. This question will be asked, and you need to answer the default, which is no.

If you are installing everything in /usr/local, you may want to try the command:

```
$ ./Configure -S
```

This will take the defaults that are already in the makefile.sh preconfigured files and generate the makefiles. This was not successful with the software on the accompanying CD since they were not in this directory. Therefore, a simple ./Configure was the command I had to execute and then answer some questions.

Once you have finished Configure, issue the command:

```
$ make install
$ make clean
```

This will generate the appropriate perl executables, including utilities to convert awk and sed scripts to perl scripts, and will install them. perl runs on many platforms, and you shouldn't have any problems with the installation.

All answers given to the ./Configure tool are contained in the config.sh files in each machine's directory. View this to determine what responses to give to Configure to build these products.

8.2.3 Conclusion

This section did not introduce you to either the syntax or functionality of perl. Instead it attempted to whet your appetite for more information about perl. By providing information about its basic capabilities and functions, this section may have prodded you into further action.

The best introduction to perl is in the Nutshell Book *Programming Perl* from O'Reilly & Associates. The other document to study is the manual page distributed with perl. There are many pages of in-depth information on perl and its syntax and functionality in the pod subdirectory. Study these files before proceeding with perl.

perl is the most powerful language to emerge in a long time. It provides many functions that fourth-generation languages purport to provide but at the same time provides the power of second- and third-generation languages. perl 5.001 is on this CD. Dump it to your hard disk and have fun.

8.3 texinfo

8.3.1 Introduction

texinfo is a documentation system which delivers both on-line and paper documentation from a single input file. texinfo is written and supported by the FSF and is, therefore, a very robust and well-written product.

texinfo files, often known as info files, are distributed with most GNU products. These files are the documentation you should use to learn about and understand the product. The most common use of the info files is in conjunction with emacs. emacs macros are distributed with texinfo such that you can view info files from within emacs and use emacs as your on-line documentation system. You can also generate device-independent (dvi) files for use with TeX and LaTeX pre- and postprocessor software (see Sec. 9.12.1 for more details).

There is a stand-alone tool called info that is distributed with tex-

info; it allows you to view documents interactively. See Sec. 8.3.3 for more details.

Finally, texinfo will eventually be merged into the emacs distribution and will disappear as an individual distribution. Keep this in mind if you plan to use texinfo and are not a big emacs user.

texinfo is a GNU product and as such is subject to its GNU General Public License as included both in the product distribution and in App. C of this book.

8.3.2 Installation

The basic installation of texinfo is similar to many other tools from GNU:

```
$ configure --prefix=/usr/local
```

This will create the proper Makefile. From this you can run make using a syntax like:

```
$ make CC=gcc install
$ make install
```

The make will generate several executables in all subdirectories. It will install a variety of LISP macro files and build several programs, including texi2dvi and makeinfo, which reformat TeX input files to dvi files and create info files from text files, respectively. These programs and especially the macros can be used by emacs and TeX to generate display output and formatted printable output from the same texinfo document.

The README file contains a listing of files in the distribution and their contents. The most interesting is the texinfo.texi file. This contains all the documentation for texinfo itself and can be viewed either with emacs or the stand-alone program info.

The other file of interest is the NEWS file. This contains a variety of information on changes and enhancements to the texinfo product for the last few versions. Take a look at this before you begin using the product.

8.3.3 Usage

texinfo consists of a variety of different types of files. The basic texinfo input file has a .texi or .texinfo suffix. Usually the subject of the texinfo document is the prefix. For example, the texinfo document for gawk is entitled gawk.texinfo. The most comon way of using texinfo and texinfo files is with emacs. You can format and display a texinfo file with a few

simple keystrokes within emacs. You can also format and display texinfo files with the info subsystem that is distributed with this book. Finally, you can use TeX to format the texinfo files for printed output.

There are also several scripts distributed with texinfo 3.1, including texi2dvi, which takes texinfo input and generates a dvi file for later TeX processing.

This distribution also contains several LISP files which are used by emacs to format a texinfo file. There are a couple of utility files which are not contained in the ./lisp subdirectory with emacs 19.17. To take advantage of these, simply move them into the ./lisp subdirectory and begin to use them.

Finally, there are makeinfo and info files which are executables which create an info file from a texinfo files and view them.

makeinfo. The makeinfo application takes texinfo input files and generates info output files which can be viewed with the info viewing system. The basic syntax is:

```
makeinfo [options...] file...
```

where options consist of:

`-Idir`	Specifies dir as a directory to search for @/include files
`-Dvar`	Defines a variable
`-Uvar`	Undefines a variable
`--no-validate`	Suppresses cross-reference validation
`--no-warn`	Suppresses warnings
`--no-split`	Doesn't split large files
`--no-headers`	Doesn't generates Node: headers
`--verbose`	Verbose option
`--version`	Displays makeinfo version
`--output file`	Specifies output filename other than the once specified in the texinfo input file with the @setfilename command
`--paragraph-indent num`	Specifies the default paragraph indent as num
`file..`	One or more files to process

You simply move to the directory where the .texinfo or .texi input file is and issue the makeinfo command. A simple example is:

```
$ makeinfo gawk.texi
```

This will generate a gawk.info file which can then be viewed with the info viewer. If you are in a directory other than the one in which the file is being built, you can use the -I option to specify a directory for any include files.

info. Once you have generated an info file, you can view it with the info command. The basic syntax is:

```
info [options...] menu-item...
```

where options are:

`--directory dir`	Specifies a directory to search for input files. You can specify a default search path with the INFOPATH environmental variable.
`-f filename`	Specifies a file to use as the initial file (default is dir).
`-n node`	Specifies a node within the input file to move to.
`-o file`	Sends output to file instead of to the default standard output.
`-h`	Produces help.
`--version`	Displays info version.
`menu-item`	Specifies a menu item within the info display file.

Info is essentially a very limited version of emacs which provides you a small set of commands with which you can manipulate the viewing of an info file. The basic command set is:

`f`	Follows a cross-reference
`h`	Invokes tutorial
`?`	Gets summary of info commands
`h`	Selects a node
`Ctrl-g`	Aborts the current command
`Ctrl-l`	Refreshes the screen
`m`	Specifies a menu item by name
`n`	Moves to the next node
`p`	Moves to the previous node
`u`	Moves up a node
`Space`	Scrolls forward one page
`Delete`	Scrolls backward one page
`b`	Goes to beginning of current node
`q`	Quits
`g`	Moves to a specified nodename
`s`	Searches for a specified string
`ESC-x print-node`	Prints the current node to lpr

The basic screen is shown in Fig. 8.2. The window in the figure is the result of the command:

```
$ info -f info.info
```

in the ./info subdirectory of the texinfo distribution. From here you can execute any of the commands above to move around within the info file.

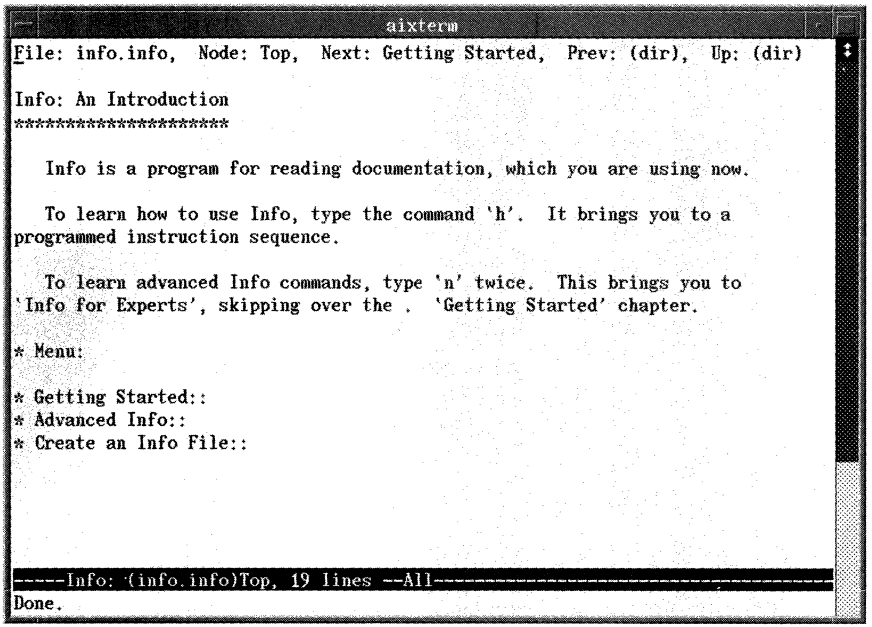

Figure 8.2 Example info window.

texinfo syntax. texinfo syntax is not the same as TeX; it replaces TeX to create a file which can be used by both info and TeX to create output. The basic syntax for a texinfo file is commands preceded by a @. A short sample texinfo file is included in the texinfo 3.1 distribution under info. For an example of this, use the info system to examine the texi.info file. This has a short example which explains how you would construct a texinfo file. From this you can create almost any kind of file you would like. The texinfo files contained in the texinfo distribution contain an example of virtually every kind of texinfo command. See these for more details on the actual language.

To get more information on the texinfo system, type the following commands when sitting in the top directory of the texinfo 3.1 distribution:

```
$ cd info
$ ./info -f info
```

Once you have played with this and learned a little about the info system, you can use info to get more information on texinfo itself with the commands:

```
$ cd ..
$ ./info/info -f texi
```

Note that some of this information is in the INTRODUCTION; however, the documentation for above command in the INTRODUCTION file is incorrect. You should use the above command to get into the info system and get more information about texinfo.

Finally, to get detailed information about the info system, type:

```
$ ./info/info -f info-stnd
```

Note that the default input filetype is .info for the info system, and it is therefore redundant to include the file suffix.

You can create a printed document that contains most of the information in the texinfo files with the TeX system. See Sec. 9.1.22 and the INTRODUCTION file in the texinfo distribution for more information.

The other way to use this is from within emacs. texinfo is an emacs mode and is invoked from within emacs with the command:

```
C-h i
```

This drops you into the info system, which is a hypertext-based system providing access to information and the ability to move around within the documents as well as return to a previous button. By placing all texinfo files in the ./emacs/info directory, you can access these files transparently through the above command. Note that you have to also modify the dir file in the ./emacs/info subdirectory to make the new info files known to emacs.

You can also access a texinfo file through emacs in a directory other than the default texinfo emacs directory by executing the command:

```
g(filename)nodename<return>
```

This means that you can move to an info file from within emacs by simply depressing the g followed by a filename enclosed in parentheses followed by a nodename within the file. For example, you can move to the gawk info file by typing:

```
g(/usr/local/gawk-2.15.4/gawk.info)
```

This will produce a screen which looks like that shown in Fig. 8.3.

You can make anything the default upon invocation of the emacs info mode by placing the appropriate info files into the ./info subdirectory in emacs and editing the dir file in the same directory to contain the nodename for the new info files.

Most software distributions from GNU come with texinfo files, and they are very useful when learning a new product. See the help for texinfo within emacs for more information.

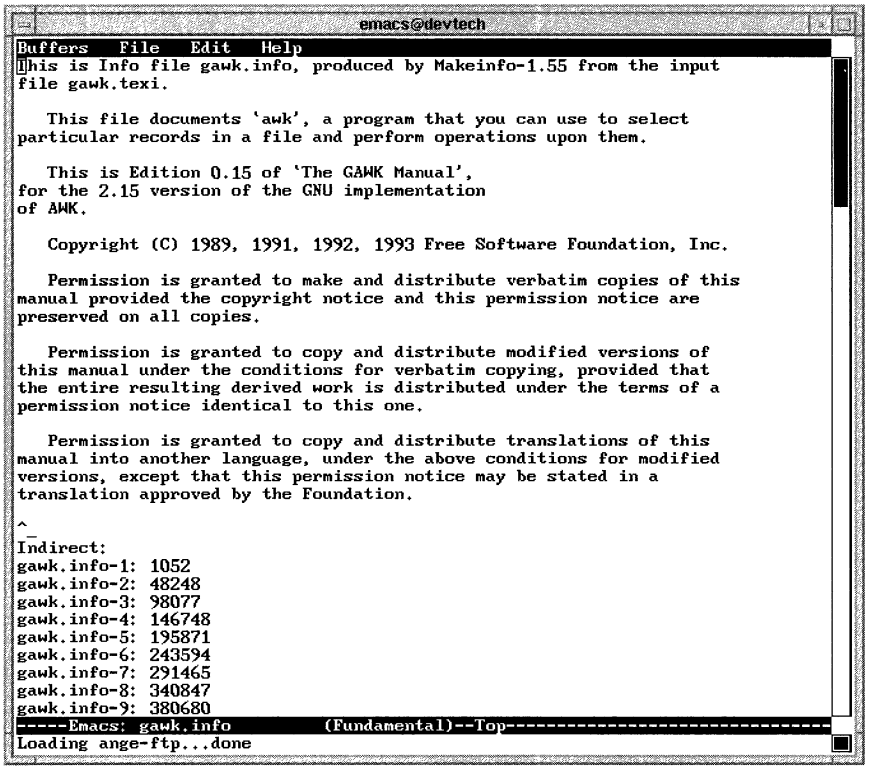

Figure 8.3 gawk info screen.

8.3.4 Conclusion

There is a tremendous amount of information contained within the texinfo files distributed with many free software products on the Internet. texinfo has its own syntax and methods for constructing files. It is beyond the scope of this book to provide information on the syntax of the language; however, all the information you need is contained in the info files in the texinfo distribution on the CD that comes with this book. By taking some time and reviewing this information using the info system described above, you will very quickly become a power texinfo user.

emacs is a very powerful editor, and texinfo is one of the reasons why. By providing a hypertext-like system from the same document from which you produce hard-copy documentation, you can save a significant amount of devloper time and cost. Because of the power of texinfo, you can use on-line documentation in a way that really enhances your productivity and effectiveness on a given system.

Finally, texinfo files are written in a format similar to TeX. In fact,

TeX is what you use to produce the texinfo in a printed format. See documentation on TeX for more information.

8.4 bsplit

8.4.1 Introduction

bsplit comes as a shell archive and works similarly to split. However, it provides a split which is common among all different platforms. The other key difference is that bsplit works well on binary files as well as on ASCII files.

8.4.2 Usage

The basic syntax of bsplit is:

```
bsplit [-size][file [prefix]]
```

where -size specifies the size of the file to split out.
file specifies a file to split.
prefix specifies a prefix to add to the output files. The default is x.

This command is very similar to split except for the binary file support. A simple example is:

```
$ ls -l elvis-1.7.tar
... elvis-1.7.tar ..... 227040
$ bsplit elvis-1.7.tar
$ ls -l
...elvis-1.7.tar ...227040
...xaa ... 50000
...xab ... 50000
...xac ... 50000
...xad ... 50000
...xae ... 27040
```

Note that the default split file size is 50,000 bytes and the file naming convention is to begin with x and cycle up from aa. You can change both the default file size and the prefix by simply including these on the bsplit command line as documented above.

To put the split files back together again, use a command like:

```
$ cat x* > newfile
```

This will take all the x files and concatenate them, in order, back into the file newfile. From here you can treat it just as you would have before you split it. In this case, you could issue the tar comand against it and should notice no differences.

8.4.3 Installation

bsplit comes as a shell archive. This means that you remove all text in the bsplit file up to the line:

```
#! /bin/sh
```

Once you create this file bsplit.c, you simply compile the file with the command:

```
$ cc -o bsplit bsplit.c
```

This creates an executable bsplit which can then be executed just as the split command is.

8.4.3 Conclusion

The bsplit command is used by many Internet utilities to split large binary files to allow for better distribution. You can use the cat command with the append option to recreate the original file as it existed before the bsplit command was executed.

bsplit is the command many people use to split software on the Internet, so you can't go wrong using it yourself.

8.5 less

8.5.1 Introduction

less is a pager. A pager controls the output of information to the screen. The most common example of a pager is the more command. There is also a pg command; however, this is not as commonly used in the UNIX environment.

less is a pager much like more, but it allows both forward and backward movement through a file. less has a command structure very similar to that of more and, therefore, is very easy for most people to use. Clearly, the biggest advantage of less is the ability to move backward through a file; however, because of its design, less does not have to read an entire file into memory like more and, therefore, comes up much faster. Many advanced UNIX users use the less pager instead of the other vendor-delivered pager tools.

8.5.2 Usage

The basic syntax of the less command is:

```
less [-[+]?aBCcdEefgGHiIMmNnqQrSsuUVwX] [-bn] [-hn] [-jn] [-kfile]
[-[oO]file] [-p pattern] [-P prompt] [-t tag] [-Ttagfile] [-x tag]
[-y tag] [-[z] tag]
[+[+]cmd] [file] ...
```

where +—a + in front of an option tells less to execute this command before entering the file.
-? displays a list of commands.
-a begins search *after* last line on current screen.
-B disables automatic allocation of additional buffers for old data storage.
-bn uses n buffers instead of the default 10.
-C clears screen, then repaints.
-c repaints screen.
-d suppresses all error messages due to lack of terminal capability.
-E automatically exits less at first end of file.
-e automatically exits less at second end of file.
-f opens nonregular file.
-g highlights only the string matched in the last search command instead of all found.
-G suppresses all string highlighting.
-hn—you can scroll n lines backward.
-I is like -i but ignores case.
-i ignores case for searches.
-jn specifies n as the line number where target results are placed (e.g., search matches).
-kfile—file is a lesskey file which contains key mappings.
-M is verbose mode.
-m is verbose mode.
-N displays line numbers.
-n supresses line numbers.
-Ofile copies output to file as well as displays, overwriting existing file.
-ofile copies output to file as well as display.
-P prompt changes prompt.
-p pattern starts at first matching pattern in file upon less invocation.
-Q is totally quiet mode.
-q is quiet mode.
-r displays raw control characters.
-S doesn't wrap lines.
-s compresses consecutive blank lines into one blank line.
-t tag is related to the ctags command.
-Ttagfile specifies a tagfile for -ttag.
-u treats backspaces and carriage returns as printable characters.
-U treats backspaces and carriage returns as control characters.
-V displays version.

-w uses blank lines to represent anything beyond the end of file.
-x disables termcap initialization of the terminal.
-xn sets tabs every n characters.
-yn scrolls forward a maximum of n lines.
-[z]n sets default window scrolling size to n lines.
file is file to page through.

You can define any options using the environmental variable LESS. This is examined at every invocation of less.

Once you have entered a file and are using the less pager, you can issue a variety of commands, some of which are listed below:

=	Displays information about the current file
! command	Shell escape
SPACE	Scrolls forward one screen
h	Displays help summary
RETURN	Scrolls forward one line
d	Scrolls forward one line
b	Scrolls backward one line
v	Invokes an editor on the current file
w	Sets window size
y	Scrolls backward one line and resizes window as necessary
R	Repaints the screen
g	Goes to line n (default of 1)
G	Goes to line n (default of end of file)
p	Goes to percent n in the file
/string	Goes to the first matching occurrence of string
?string	Goes backward to the first matching occurrence of string
n	Repeats previous search
N	Repeats previous search but in opposite direction
:e [file]	Examines new file
:n	Examines next file in filelist from command line
:p	Examines previous file in the filelist from the command line

Almost all of the above commands can be followed by an integer which represents the number of lines on which to act. Most of the commands have a default value of 1.

You can use a program called lesskey to define all keys you may wish to use with less. Keep in mind that this will change the mapping for your keys, and the fundamental behavior of less may change depending on the redefinition of your keyboard. lesskey is beyond the scope of this book. See the man page less.man and lesskey.man for more information.

8.5.3 Installation

You can install less using the standard methodology; however, it is not required with the code as delivered. To build without using configure, type:

```
$ make prefix=/usr/local
```

If you want to use gcc instead of CC, simply type:

```
$ make CC=gcc prefix=/usr/local
```

Then install with the command:

```
$ make install
```

If you have problems, examine the makefile for potential problems with your system. The build is fairly straightforward, and you shouldn't have any real problems with it. If you do, there is a mailing address in the README file of the distribution.

8.5.4 Conclusions

less is a very powerful tool which allows you to examine a file and move back and forth within that file with ease. You can also move into an editing function if you so desire. Through the use of keypad definitions (lesskey) and environmental variables, you can modify less's behavior to match your specific needs. See the man pages included with this distribution for more details.

8.6 bash

8.6.1 Introduction

bash is the GNU Bourne-Again Shell and is a sh-compatible shell interpreter which provides significantly enhanced functionality over sh (Bourne shell), csh (C shell), and ksh (Korn shell). While sh and csh are provided on most UNIX platforms, ksh is not and therefore is not as widely used as the other two shells.

You can purchase ksh from AT&T for use on any system; however, you may not be willing to provide additional funding for a shell for your version of UNIX and, therefore, are limited to sh and csh. ksh has more functionality than both sh and csh do and, in fact, combines the best functionality from both sh and csh in its implementation. But because of its lack of general availability because of its cost, there is a need for additional shell functionality that bash can provide.

bash is intended to be POSIX compliant, which provides consistent and documented interfaces and command structures as well as heterogeneous support of UNIX platforms. This is important as you get larger and more heterogeneous environments and need to support and use systems with diverse UNIX flavors. bash utilizes the best functions from sh, csh, and ksh and provides significant functionality above and beyond them, and is free.

bash is a GNU product and as such is subject to its GNU General Public License as included both in the product distribution and in App. C of this book.

8.6.2 Usage

The basic syntax of bash is as follows:

```
bash [options] [file]
```

where options consist of both internal (built-in) and external options which are documented in some detail below:

`-, --`	Disables further command line option processing. All strings following - or -- are filenames or arguments.
`-c commands`	Reads commands from commands.
`-i`	Makes shell interactive.
`-login`	Invokes bash as if invoked by login.
`-nobraceexpansion`	Doesn't perform curly brace expansion.
`-noprofile`	Doesn't load /etc/profile or .bash_profile.
`-norc`	Doesn't load personal initialization file .bashrc.
`--quiet`	Doesn't display start-up information.
`-rcfile file`	Uses file as personal initialization file instead of .bashrc.
`-s`	Reads commands from standard input.
`-version`	Displays current version of bash.
`file`	File containing shell commands to be executed on bash invocation.

Shell programming. This section will not cover all the functionality provided by bash; however, some discussion of the shell programming characteristics and built-in commands is necessary to understand the context of the shell and its capabilities.

Many of the same functions supported in the other shells are supported in bash. Things like reserved words, pipes, metacharacters, and other similar functions are fully supported. In fact, making the assumption that you can use sh programming syntax is a good assumption and will create the proper framework for developing shell scripts and executing commands.

There are ways to control the execution of shell programs and com-

mands with basic syntax provided with bash. The standard throughout the bash command documentation is the use of the word list to represent a series of one or more commands separated by semicolons. An example is:

```
list1 = (command1;command2; command3; ...)
```

where commandn is simply a regular UNIX command or internal shell command. As with sh, this allows you to create multiple commands on the same command line. This is a powerful feature for both interactive use and scripting.

Some examples of command syntax are:

`#`	In first column, denotes a comment
`(command;command;...)`	Executes all given commands in a single subshell
`{command;command;...}`	Executes all given commands in current shell
`for name [in word;] do list; done`	Standard loop syntax
`case word in [patttern [|pattern]...list;;] ...esac`	Standard case syntax
`if list then list [elif list then list] ...[else list] fi`	Standard if command syntax
`while list do list done`	Standard while syntax
`until list do list done`	Somewhat unique until command syntax

There are metacharacters which mean special things to the bash interpreter. They are:

```
| & ; ( ) < > <space> <tab>
```

If you want to physically represent the shell metacharacters on a command line, you must "escape" them much as you have to in the other shells. In other words, if you have a string:

```
kevin()
```

which you want to represent on the command line as a literal and not have bash interpret the () to mean something special, you would use:

```
kevin\(\)
```

where the \ (backslash) escapes the metacharacters from interpretation.

There are two other ways to affect command line interpretation: the single quote and the double quote. They behave differently, and therefore it is important to understand how they work in bash. The single

quote maintains the literal string with no command substitution or shell interpretation. For example:

```
'string1;$string2;!string3'
```

will present the literal string as displayed above as the result of the command interpretation. The only exception to this is the single quote itself, which may not be placed within a pair of single quotes.

Double quotes maintain the literal string with the exception of $, ‘, and \ interpretation. This is useful to provide literal metacharacters other than those listed. The same string as above:

```
"string1;$string2;!string3"
```

will provide the output:

```
string;someotherstring;!string3
```

where someotherstring is the interpreted value of string2. Note that this provides basic string-level substitution without interfering with your ability to pass most literal strings directly to the program invoked from the shell.

This brings us to a discussion of shell variables and parameters. The basic syntax for viewing and setting variable values is:

```
name=[value]
```

where name is the name of the variable to set.
value - is the value to set for name. If value is not set, the default is null.

Simple display of the value of name is with the metacharacter $. For example, to set the value of the variable of author to kevin and view it, use:

```
$ author=kevin
$ echo $author
author=kevin
```

There are special predefined variables in bash which are very similar to sh and ksh. They are:

BASH	Full pathname of current bash invocation.
CDPATH	Searches path for cd command.
ENV	If set, provides bash initialization file.
EUID	Effective user id.
HISTFILE	File to save history command.

HISTSIZE	Size of history recall list.
HOSTTYPE	Automatically set to current system type.
IGNOREEOF	If set, specifies the number of EOFs that must be received before bash exits. The default value is 10.
LINENO	Current line number in a script.
MAIL	Set to a specific filename; bash notifies you when you receive mail to this file.
MAILCHECK	Sets time in seconds to check for new mail.
MAILPATH	Specifies a path to check for new mail. Default is system mail area.
OLDPWD	Previous working directory.
OPTARG	Last option argument processed by built-in command.
OPTIND	Last option index processed by built-in command.
PPID	Parent process id.
PWD	Current working directory.
RANDOM	Each time RANDOM is used, it generates a random number.
SECONDS	Equates to the number of seconds since bash invocation.
SHLVL	Incremented each time bash is invoked.
TMOUT	Time in seconds to wait for terminal input before timing out.
UID	Current user id.
command_oriented_history	Attempts to save multiple command line commands as a single history entry for later easier use.
noclobber	Will not overwrite existing files with redirection commands.
nolinks	Will not follow symbolic link paths for command execution.
notify	Notifies job control issues immediately.

There are many other variables in bash, but these are the most commonly used ones. See the documentation for more details, specifically the man page delivered with bash (bash.1).

Many of the things you have come to expect from a shell interpreter in UNIX are available and supported in bash. Things like redirection, command line expansion and substitution, built-in commands, aliases, and job control are fully supported. Many of these functions, such as job control and aliases, operate much the same as csh, while functions such as redirection and substitution operate much as sh does.

There are a variety of ways you can structure the command line within bash. There are also a variety of ways you can define bash to interact with respect to command line editing and terminal emulation. Much of the functionality of bash is derived from emacs, and therefore commands are documented in a similar fashion to emacs. For example, the syntax C-c means to depress the CTRL key and the lowercase c key at the same time. The documentation delivered with bash uses syntax

like M-c to denote pressing the ESC key and the lowercase c key in sequence. To be consistent this section will use the same syntax.

Some of the more basic bash capabilities and commands are:

`set editing-mode vi`	Establishes command line editing with vi commands (similar to ksh)
`C-a`	Moves to beginning of line
`C-b`	Moves back one character
`C-d`	Deletes text under cursor
`C-e`	Moves to end of line
`C-f`	Moves forward one character
`C-k`	Removes text following cursor
`C-l`	Refreshes the screen
`C-n`	Recalls the next command from the history list
`C-p`	Recalls the previous command from the history list
`C-r`	Moves sequentially through the history list in reverse order
`C-s`	Moves sequentially through the history list in forward order
`C-u`	Kills the current line
`C-v`	Adds character following C-v as a literal (good way to enter ESC, etc.)
`M-/`	Attempts filename completion
`M-[td]`	Attempts userid completion
`M-<`	Moves to the first command in the history list
`M->`	Moves to the last line in the history list
`M-b`	Moves back one word
`M-c`	Capitalizes first character of current word
`M-d`	Removes text in the current word following the cursor
`M-f`	Moves forward one word
`M-l`	Lowercases the current word
`M-u`	Uppercases the current word

All of the above keystroke sequences can be executed at the command line to provide the documented functionality. bash has by far the most flexible and powerful user interface for a shell since it combines all the best from sh, csh, and ksh. The above listing is a sample of that functionality. See the accompanying documentation for more details.

Most of the built-in commands are similar to those of the other shells. Commands such as echo, eal, exec, fg, jobs, popd, pushd, set, test, trap, ulimit, umask, and wait are fully supported. There are some unique commands which deserve specific mention. Some of these are:

`builtin [shell-builtin [args]`	Executes shell-builtin with args. This is used to redefine a built-in command for execution of itself.
`dirs [-l]`	Lists directories in current remembered directories created and manipulated by pushd and popd.
`help [pattern]`	Provides bash-specific help on built-in commands or those matching pattern.

`times`	Displays user and system times since shell invocation.
`ulimit [-HSacdfmstpn [limit]]`	Displays and manipulates process resources: -H—specifies a hard limit on the resource, which says that it cannot be increased at a later time. -S—specifies a soft limit on the resource, which says that it can be increased to the hard limit at a later time. -a—displays all current limits. -c—core file maximum size. -d—data segment maximum size. -f—maximum file size. -m—maximum resident size. -s—maximum stack size. -t—maximum CPU time. -p—pipe size. -n—number of open file descriptors. limit—specifies a limit to set. If limit is not specified, all above options cause the current variable value to be displayed.

Some systems may not support the ulimit command. Try it on your system and see.

In the documentation subdirectory, there are both nroff input pages and Postscript files which describe bash in great detail. Print out the bash.ps and bash_builtins.ps files for much more detailed information on bash.

8.6.3 Installation

There are a variety of files associated with bash and its installation and usage. One of the first to examine is the ChangeLog file. This file contains information which describes changes to the current version. This is important since bash is undergoing constant development and change, and new features and bug fixes will be documented here.

bash does not use the configure subsystem to assist with the build; however, it does have the capability to determine which machine you are on based on information and logic placed in the makefile itself. While this may not always be accurate, it does provide a good first guess as to machine type and the files necessary for the correct build to occur.

The basic syntax to build bash is:

```
$ make clean
$ make CC=gcc
```

Once you have done this, the documentation recommends you attempt to run bash with the command:

```
./bash
```

This should invoke an interactive bash shell. If it does not, see the file INSTALL for information regarding specific machine information.

There are a variety of product and machine-specific issues which may need to be addressed when you build bash. The file machines.h contains both a general macro definition section and a specific section for a variety of machine types. Look in this file for your machine type and ensure that the build agrees that this is the machine type you are using. Once you have invoked the make command, it creates a file named .machine which tells you what type of machine it thought you were using.

If you are using tools other than the standard ones shipped with your machine, you may need to edit the Makefile to use these tools. For example, if you are using the GNU C compiler, you should uncomment the related lines in the beginning of the cpp-Makefile file. You should also uncomment the bison line in the same file since the yacc compiler may have problems building bash. So install bison before you install yacc.

For example, it was necessary to change the prototype definitions for both rindex and index in general.c to avoid a prototype conflict on the RS/6000 using gcc. These are the kind of things which sometimes occur when building these types of products. See the documentation with bash for more details on building and using bash.

When make executes, it displays what it thinks is the current machine type and stores it in a file named .machine in the current build directory. Check this file to see what machine type bash was built for if you don't see it displayed on your screen. See also the file machines.h, which contains information regarding compilation parameters for all defined machine types. In fact, if you are porting to a machine that is not in machines.h, you should create your own block of information for your particular architecture and make based on these parameters.

Once you have successfully built bash as outlined above, you will need to install the bash executable. The basic command to do this is:

```
$ make install bindir=/usr/local/bin
```

Note that this is different from that noted in the INSTALL file, but this is the correct command to install bash poperly.

Bug reports should be mailed to bash-maintainers@ai.mit.edu. See the file INSTALL for more information on bug reporting.

8.6.4 Conclusion

bash is a very powerful tool which provides shell functionality from sh, ksh, and csh as well as some of its own unique functionality. Since this functionality is free and it gives you the ability to run the same shell on various diverse UNIX platforms, you can significantly enhance the portability of your environment while at the same time reducing costs

for your environment. bash is backward compatible with sh, csh, and ksh and will support most shell scripts, commands, and functional subsystems to ensure protection of your current investments.

Much of this documentation was taken directly from the GNU bash man page. See this for more information. Note also that, as with most newer GNU products, there is a texinfo file in the documentation subdirectory which you can view with info or emacs to get more information than is in the man page. There are also Postscript format files in the documentation subdirectory which can significantly enhance your understanding of the bash product.

8.7 diff

8.7.1 Introduction

GNU diff is a package which provides GNU diff, diff3, sdiff, and cmp programs. These programs provide significant enhancement to the standard diff you get with most UNIX operating systems. This tool is also required for other GNU and free software tools such as CVS; therefore, it is important to understand its capabilities.

diff provides a facility which generates files listing differences between files. You can then use these files to provide upgrades and other difference files which are used by utilities such as patch to apply these differences.

8.7.2 Installation

You can execute the standard GNU configure script, which you can run as follows:

```
$ configure --prefix=/usr/local
```

This will build the appropriate Makefile, which can then be executed with the command:

```
$ make
```

This will generate the appropriate executables in the current directory structure. Once you have ensured that the appropriate executables were built, you need to install the diff executables with the command:

```
$ make install
```

This will place the diff executables in the prefix/bin directory. Remember that you defined prefix when you issued the configure command. In this case they would be placed in /usr/local/bin.

This is all there is to building and installing diff.

8.7.3 Usage

The GNU diff product consists of several binaries: cmp, diff, diff3, and sdiff. Each of these will be described below in enough detail to make them usable. There are no accompanying man pages or documentation except for info files. This means that you may want to view the info files with the info system for more details on the diff utilities and associated syntax. See Sec. 8.3 for more details.

GNU diff provides differences between two files or corresponding files in different directories. Along with the diff command, GNU diff provides diff3, cmp, and sdiff commands, which provide different functional aspects from diff. Each is described below.

diff is often used to distribute patch files which can then be applied with the patch command to update and/or change current software distribution files.

diff automatically checks the format of the file and if it thinks the files are binary, it will simply report that it found differences. It cannot differentiate and locate the difference but will simply note that they exist. There are other utilities which are more useful for binary comparisons, but diff will support this notion at a base level.

diff. The basic syntax for diff is:

```
diff OPTIONS input-file output-file
```

where OPTIONS consists of:

`-a, --text`	Treats all files as text.
`-b`	Ignores changes in blanks and tabs.
`-B`	Ignores changes that insert or delete blank lines.
`--brief`	Reports only that the files differ without any details.
`-c`	Context output.
`-C LINES, --context[=LINES]`	Shows LINES number of lines around the differences reported.
`-D NAME`	Makes merged #ifdef format output.
`-e, --ed`	The output is a valid ed script.
`-H, --speed-large-files`	Applies heuristics to increase the speed of the search on large files.
`-i`	Ignores case.
`-I REGEXP`	Ignores changes that insert or delete lines matching REGEXP.
`--ignore-all-space`	Ignores all whitespace.
`--ignore-case`	Ignores case.
`-l,--paginate`	Passes output through pr to paginate it.
`--left-column`	Displays only the left column on output.
`-n,--rcs`	Outputs RCS format diffs.

`-N, --new-file`	When performing directory comparisons, if a file is found in only one directory, it treats it as present but empty in the other directory.
`-S, --report-identical-files`	Reports when files are identical.
`--sdiff-merge-assist`	Displays additional information that sdiff can use.
`--show-c-function`	Displays the C function each change is in.
`--side-by-side`	Uses the side-by-side output format.
`-t, --expand-tabs`	Expands tabs to spaces.
`-u`	Displays with unified format.
`-v, --version`	Displays current diff version.
`-w`	Ignores horizontal whitespace when comparing lines.
`-W COLUMNS, --width=COLUMNS`	Specifies the width of columns to use in the side-to-side format.

Context refers to displaying lines around the difference lines reported. This is useful in providing information about the context of the change.

There are also ways to control the format of the output. See the info files for more details, particularly the node Output Formats in the info files.

Some simple examples of using diff are illustrated below. Using a file test.c which contains:

```
#include <stdio.h>

main() {
printf("This is a test"\n);
}
```

and a file named test.new.c which contains:

```
#include <stdio.h>

main() {
printf("This is a new test"\n);
}
```

you can issue commands like:

```
$ diff test.c test.new.c
4c4
< printf("This is a test"\n):
---
> printf("This is a new test"\n):
```

The arrows indicate the first file or input file (<) and the second file or output file (>). Another example using the context option is:

```
$ cmp -C 3 test.c test.new.c
*** test.c Wed Jun 14 14:03:17 1995
--- test.new.c Wed Jun 14 14:03:38 1994
***************
```

```
*** 1,5 ****
#include <stdio.h>

main () {
! printf("This is a test"\n):
}
--- 1,5 ----
#include <stdio.h>

main () {
! printf("This is a new test"\n):
}
```

This outlines the context format which provides more information for the results of the diff command.

cmp. The cmp command shows offsets and line numbers for the input files. cmp can also show character-for-character differences, whereas diff functions on a line level only.

The basic syntax for cmp is:

```
cmp OPTIONS input-file [output-file]
```

where OPTIONS are:

`--print-chars`	Displays the differing characters
`-l, --verbose`	Displays the decimal offsets and octal values of all differing bytes
`-s, --quiet`	Silent mode

The input-file is the input file to be operated on, and the output-file is the optional output file. The default is standard output.

A simple example of cmp is as follows. Using a file test.c which contains:

```
#include <stdio.h>

main() {
printf("This is a test"\n);
}
```

and a file named test.new.c which contains:

```
#include <stdio.h>

main() {
printf("This is a new test"\n);
}
```

invoke the cmp command as follows:

```
$cmp test.c test.new.c
test.c test.new.c differ: char 50, line 4
```

```
$ cmp test.c test.new.c
test.c test.new.c differ: char 50, line 4 is 164 t 156 n
```

These are simple examples of using cmp. Clearly you can work on larger and more sophisticated files with it without any difficulties.

diff3. diff3 shows differences between three files. This is commonly used for software development where more than one person is making changes to a file simultaneously and you want to see all differences from the original. diff3 can also merge the difference files and flag any possible conflict in the merged file. This is extremely useful for support of environments such as parallel build.

The basic syntax for diff3 is:

```
diff3 OPTIONS file1 file2 file3
```

where OPTIONS are:

`-a`	Assumes all files are text.
`-A`	Merges all changes from file2 to file3 into file1.
`-e, --ed`	Creates an ed script which merges all changes from file2 and file3 into file1.
`-i`	Creates w and q commands at the end of the ed script to maintain System V compatibility. This option is useful with -AeExX3 options.
`-m, --merge`	Applies the edit script to file1 and sends results to standard output.
`--show-all`	Incorporates all unmerged changes from file2 and file3 into file1, surrounding all overlapping changes with bracket lines.
`-v, --version`	Display diff3 version.
`-x`	Like -e except outputs only the overlapping changes.

file1, file2, file3 are input files in which file1 may be modified depending on the command OPTIONS you choose.

There are other diff3 options; see the info pages for more details.

sdiff. You can use sdiff to merge files interactively. The basic syntax is:

```
sdiff -o OUTFILE OPTIONS input-file output-file
```

where OUTFILE is the output file created from the merge.
OPTIONS consist of:

-a, -text—treats all files as text files

-b, --ignore-all-space—ignores changes in the amount of whitespace

-B, --ignore-blank-lines—ignores changes that just insert or delete blank lines

-H, --speed-large-files—uses heuristics to increase performance on large files

-i, --ignore-case—ignores case
-I REGEXP, --ignore-matching-lines=REGEXP—ignores changes that match REGEXP
-l, --left-column—displays on the left column of a side-by-side output
-t—expands tabs to spaces
-v, --version—displays version of sdiff
-w COLUMNS, --width=COLUMNS—displays output columns of width WIDTH
input-file is the file from which sdiff gets changes.
output-file is file on which sdiff merges input-file.

In other words, sdiff takes all options and merges input-file with output-file and creates OUTFILE. A simple example is:

```
$ sdiff test.c test.new.c
#include <stdio.h> #include <stdio.h>

main () { main () {
printf("This is a test"\n): | printf("This is a new test"\n):
}       }
```

This shows the default without a -o option, which is a side-by-side output. If you specify the -o option, you get a following file.

```
$ sdiff -o new.c test.c test.new.c
#include <stdio.h> #include <stdio.h>

main () { main () {
        printf("This is a test"\n): | printf("This is a new test"\n):
%?
l:      use the left version
r:      use the right version
e l:    edit then use the left version
e r:    edit then use the right version
e b:    edit then use the left and right versions concatenated
e:      edit a new version
s:      silently include common lines
v:      verbosely include common lines
q:      quit
```

Note that what happens is that you are dropped into an interactive editing subsystem which has a prompt of %. From this point you can type a ?, which will display a list of available commands. You can then choose to edit or simply choose one version of a file over the other. Once you have finished choosing and/or editing the appropriate lines, your resultant file will be contained in new.c. This is a very powerful interactive merge system which can assist you with supporting environments where more than one person is modifying a file at the same time. Sys-

tems like RCS and other tools make use of this for their file merge systems.

You can control which editor is invoked by sdiff by setting the EDITOR environment variable before invoking sdiff.

8.7.4 Conclusion

While most UNIX machines come with a diff command, GNU diff adds significant functionality to the base diff offerings. It also provides a consistent diff interface and syntax across a variety of platforms. Finally, it is used to distribute many product patches and updates and will be critical in the use of several other GNU and other freeware products.

8.8 screen

8.8.1 Introduction

screen is a utility which provides a full-screen window manager that allows a dumb terminal to support multiple terminal sessions. Full vt100 support is included, and there are history buffers and copy and paste capabilities between window sessions. This is similar to the new alphawindows protocol which is coming into vogue; however, this is free and runs on dumb vt100-style devices.

Each screen session has a UNIX command associated with it, and as you toggle back and forth between screens, you are automatically attached to the UNIX command assigned to that window. The most common UNIX command within a window is a shell command such as /bin/csh, /bin/sh, or /bin/ksh. This provides a terminal window just as if you were logging on to the machine in a single-session mode.

screen is a GNU product and as such is subject to its GNU General Public License as included both in the product distribution and in App. C of this book.

8.8.2 Usage

The basic command line use for screen is:

```
screen [-A] [-L] [-O] [-R] [-a] [-c file] [-D|-d [pid.ty.host]] [-e
xy] [-f,-fn,-fa] [-h n] [-i] [-l,-ln] [-ls,-list] [-t name,-k name]
[-r [pid.tty.host]] [-s] [-wipe] [command [args]]
```

where -A adapts window size to display size.
-L—last character position on automargin terminal is writeable.
-O matches more closely your terminal's characteristics if it is not a vt100.

-R resumes the first detached window.
-a includes all display capabilities in each window's termcap.
-c file overrides the default configuration file from $HOME/.screenrc to file.
-D | -d [pid.tty.host] begins session as undetached.
-e xy defines the command character x and the literal command character y.
-f, -fn, -fa turns flow control on, off, or to automatic switching mode, respectively.
-h n specifies the size of the history buffer as n lines.
-i causes interrupt key to interrupt display automatically.
-l,-ln turns logon mode on or off.
-ls, -list displays a list of screen sessions.
-tname,-kname sets the title for the default shell or specified program.
-r [pid.tty.host] resumes a detached screen session.
-s sets the invocation shell to that specified in command.
-wipe is same as -ls but removes sessions instead of simply marking them as dead.
command [arg] is command to invoke with associated arguments.

Note that the environmental variable SHELL will determine which shell will be invoked by default and that the tty setting will determine the characteristics of the terminal emulation and what features are utilized and supported.

You can customize the key bindings and other features with the command:

```
C-a : command
```

where command is taken from the list described on p. 13 of the screen manual page included with the screen distribution. See this file for more details. With the customization command, you can modify almost every aspect of your terminal emulation and multiwindow environment on the fly.

Command mode. There are a variety of interactive commands you can execute which talk directly to the terminal manager. Some of the more basic are:

`C-a C-\`	Kills all windows and terminates screen.
`C-a [`	Enters copy/scrollback mode, which allows for screen cut and paste. The syntax is very similar to vi.
`C-a ]`	Writes current contents of paste buffer to the current window.

`C-a ?`	Displays a list of available commands.
`C-a n`	Where n is a number between 0 and 9 and represents the window session number.
`C-a C-a`	Switches to previously displayed number.
`C-a c`	Creates a new window with a shell and moves to this window.
`C-a C`	Clears the current screen.
`C-a d`	Detaches the current screen.
`C-a D`	Detaches the current screen and kills all associated processes.
`C-a f`	Cycles flow control from automatic to on or off.
`C-a h`	Generates a hard copy of the current screen.
`C-a H`	Begins logging of current window to hardcopy.n where n is the window number.
`C-a i`	Displays information about current window.
`C-a k`	Kills the current window and switches to previous window.
`C-a l`	Refreshes the current window.
`C-a L`	Determines whether entry in /etc/utmp is generated. This determines whether you are logged in or not.
`C-a M`	Toggles monitoring of the current window. If the window is placed in the background and monitoring is on, you will receive status upon output to that window.
`C-a n`	Switches to next window.
`C-a p`	Switches to previous window.
`C-a q`	Sends CTRL-q to the current process in the current window.
`C-a r`	Toggles automatic line wrap.
`C-a s`	Sends CTRL-s to the current process in the current window.
`C-a t`	Displays the time of date, hostname, and load averages.
`C-a v`	Displays version and generation date.
`C-a w`	Displays a list of all windows. Current window is marked with an *, and the previous window is marked with a -.
`C-a x`	Runs screenlock on the current window.
`C-a z`	Suspends all screen sessions.
`C-a Z`	Resets window to default values.

There is a message line at the bottom of the screen which keeps you informed about the status and results of an executed command.

To exit screen, you can type C-a C-\ or simply type exit at the command prompt. When you have exited the final screen, you will see the message "screen is terminating." At this point you are back at the original session from which you invoked screen.

There is an issue of flow control related to the screen system. When you want to stop the flow of information, you typically press a C-s. When you want to resume the flow, you press C-q. With screen, if you press these, screen may interpret them for you or pass them to the underlying application. By enabling or disabling flow control with the C-a f command, you can change the behavior of the screen subsystem. If you want to ensure that you send C-s and C-q to the application and bypass any processing by screen, use the C-a s and C-a q commands.

There is much more information in the 24+ page man page included with the distribution.

8.8.3 Installation

First you need to run the configure script. The typical syntax is:

```
$ ./configure --prefix=/usr/local
```

This will generate not only the appropriate Makefile but an appropriate config.h as well. You may want to preview the config.h file to ensure that your paths and other variables are correct before proceeding with the build. There is a section in config.h called the "User Configuration Section" which outlines all variables you may wish to modify. See these before compiling screen.

Once you have modified the config.h file, if necessary, you can proceed with a normal configure build just as you do with most GNU products. The typical command set to build using the configure product is:

```
$ make
$ make install
```

Keep in mind that there are more things you can do with the configure command. See Sec. 7.4 for more details.

8.8.4 Conclusion

screen is a very powerful utility which gives many users of dumb terminals some relief from the limitation of one session. By providing multiple sessions (up to nine sessions at once), you can utilize your dumb vt100-style terminal much more effectively and can dramatically increase your productivity without incurring the expense of purchasing a new windowed device such as an X-station or workstation.

8.9 fax

8.9.1 Introduction

The Netfax facility provides a distributed fax server capability which will allow you to preview, send, and receive Group 3 faxes from any machine on a LAN. The software is free and the only hardware required is a EIA-592 Asynchronous Facsimile DCE Control Standard, Service Class 2. An example given in the associated documentation is the Everfax 24/96D, which retails for $499. The main phone number for Everfax is 1-800-821-0806. There are other modems that will sat-

isfy the hardware requirements as well. See your UNIX vendor for more information.

The primary server is a process known as a fax spooler, and it runs on a machine which is accessible by all other fax client machines. By posting either mail or flat files to a spooler directory, the server will scan the queue periodically and distribute the fax material appropriately. You can receive Group 3 faxes, and they will be stored in the appropriate directories for receipt. There are programs delivered which allow you to modify the format of these files into tiff and other formats for use by other tools and utilities.

The files to be faxed support ASCII, Postscript, and TeX dvi files as input. The fax spooling subsystem will generate the appropriate Group 3 fax output automatically. This provides many tools and utilities with the ability to send faxes transparently by merely creating an output file in one of the supported fax server input formats.

This is a very powerful tool which will provide fax capabilities for very low cost.

fax is a GNU product and as such is subject to its GNU General Public License as included both in the product distribution and in App. C of this book. There is also an associated copyright from M.I.T. which is distributed with the product on the CD. See this for more information.

8.9.2 Usage

There are several commands related to the Netfax spooling system. The basic ones are:

```
fax -p phone [-c] [-h host] [-m] [-r rec] [-s send] [-S phone]
[-u user][file] faxps -p phone [-c] [-h host] [-m] [-r rec]
[-s send] [-S phone] [-u user][file]
```

where -p phone phones number of fax receiver machine (uses Hayes string formats).
-c sends fax using information in -r, -s, and -S options.
-h host specifies host acting as fax spooler.
-m sends mail to you when fax is delivered.
-r rec is name of receiver.
-s send is name of sender.
-S phone is phone number of sending fax board.
-u user uses name to queue the job instead of your local userid.
file is file to fax (format is different depending on whether you are using fax or faxps).

The fax command is the primary command to send a plain ASCII file as a fax. The faxps command takes a Postscript file as its input file and

sends it as a Group 3 fax. The input file format is the only difference between fax and faxps.

The phone number sequence is Hayes compatible. For example, to dial a fax at the number 555-1212, you would use a command like:

```
fax -p 5551212 ...
```

You can use other dial characters such as T for tone dialing and , for pause. If you are disabling call waiting on your phone, you might use a command like:

```
fax -p *70,,5551212 ...
```

This will dial *70, wait 4 seconds, and dial 5551212. A full-blown command might look something like:

```
$ fax -p *70,5551212 -c -r "John" -s "Kevin" -S "5552020" fax.txt
```

Using faxps is exactly the same syntax as fax only the input files are Postscript and not plain text as with the fax command. A sample command might look like:

```
$ faxps -p 5551212 -c -r "Joe" -s "Kevin" -S "5552040" fax.ps
```

where fax.ps is a Postscript file. faxps uses Ghostscript to convert the input file (in this case fax.ps) from Postscript to Group 3.

faxeng. faxenq is a command which queues your faxes to the fax spooler server for later distribution. This provides an asynchronous fax capability. This is especially nice to use as a fax server where you have multiple faxes to send to different locations and you want this to happen while you continue to work on other things. The syntax is:

```
faxenq phone1 phone2 ... [--- file1 file2 ...] [-m] [-u user]
[-h host]
```

where phonen is one or more phone numbers to send the fax to.
filen is one or more files to be queued to the fax spooler.
-m sends mail when the fax is sent.
-u user specifies user as the sender of the fax.
-h host specifies fax spooler server.

faxmail. faxmail allows you to send mail as a fax by transmitting standard input mail as a fax. When you enter faxmail, it asks you for all pertinent information such as sender, receiver, phone number, etc.,

which will allow it to queue it to the fax spooler server for later transmission.

faxrm. faxrm removes a fax job from the spooler queue. Once the job has started, you cannot remove the job from the queue. The syntax is:

```
faxrm job [-h host]
```

where job is job as specified in the job spooler queue.
-h host specifies spooler host.

faxspooler. faxspooler is the daemon which serves all fax activities to the network. Faxes are received by the host, and they are spooled in an incoming directory numbered sequentially. Each page of a fax is given its own subdirectory within the created directory. E-mail is sent when the fax is received. The syntax is:

```
faxspooler [-lloglevel] [-fdev] [-dmaxtime] [-rwaittime] [-D] [-Idir]
[-Odir] [-Eaddress]
```

where -lloglevel specifies loglevel as the debug log level from 0 (least verbose) to 7 (most verbose).
-fdev specifies dev as the fax board.
-dmaxtime is the maximum time to try sending a fax.
-rwaittime specifies wait time between fax retries.
-D runs the spooler as a daemon.
-Idir is incoming fax spool directory.
-Odir is outgoing fax spool directory.
-Eaddress is e-mail notification address for incoming faxes.

faxq. faxq lists all jobs in the queue on host. The syntax is:

```
faxq [-hhost]
```

where -hhost specifies the host of the master fax spooler.

These basic commands are all you need to send and monitor faxes on your LAN. With Netfax, you can provide network access to a central fax serving facility for minimal cost and maintenance.

8.9.3 Installation

Netfax comes with a standard makefile which builds fax and the associated executables. The makefile is formated to use gmake by default; however, you can use standard make if you don't have gmake installed.

The three files that may need modification are:

./include/conf.h

../include/conf.mk

./include/fax_prog.mk

These files consist of header information, a base configuration makefile, and a makefile which will build the appropriate executables, respectively. conf.h contains information pertaining to the actual device names for the fax board and commands which determine the default behavior of the fax server such as retry times, maximum retries, and the Postscript converter to use when converting from Postscript to Group 3 fax standards. Look carefully at this file before building.

The conf.mak and fax_prog.mk are template files which document the makefile features that are needed to build Netfax. These *.mk files are in subdirectories which will be automatically built when the original make is executed. Netfax requires gmake because of its use of subdirectories. To build, move the the main Netfax directories and issue the command:

```
$ gmake
```

If you don't have gmake installed, you will have to move the *.mk template makefiles into the current working directory with the main makefile before issuing the make command. It would be easier to simply take gmake from the included CD to build Netfax.

The other requirement for the best use of Netfax is Ghostscript 2.41 or greater. This provides a Postscript interface and the ability to display Postscript on a bitmapped display. See Sec. 10.1 for more information. The device to use for Ghostscript and fax is dfaxhigh. See the INSTALL file for exact details on how to build the correct fax driver for Ghostscript.

There are subdirectories containing manual pages (man), documentation files (doc), Postscript programs (ps), and others.

This system is not delivered compiled on the CD because you need to link it to your particular fax board. You will need to build it yourself. Have fun.

8.9.4 Conclusion

Netfax provides the capability of LAN fax serving at a very low cost. By providing software which allows distributed fax server management, Netfax provides your LAN with the ability to fax ASCII or Postscript files as well as TeX files.

Netfax provides most of what you can get from expensive commercial

packages without the cost or complexity. Give it a try and see how it works.

8.10 mtools

8.10.1 Introduction

mtools contains a variety of tools which allow you to manipulate DOS filesystems and disks. This gives you the ability to write, read, copy, format, and copy DOS files and disks to and from UNIX machines. This section discusses mtools 2.0.7.

8.10.2 Usage

There are several commands which come in the mtools command set. The basic commands included are mattrib, mcd, mcopy, mdel, mdir, mformat, mlabel, mmd, mrd, mread, mren, mtype, and mwrite. Each command and basic syntax is outlined below.

mattrib. The syntax is:

```
mattrib [-a|+a] [-h|+h] [-r|+r] [-s|+s] dosfile ...
```

where -a,+a removes or adds attribute bit.
-h,+h removes or adds hidden bit.
-r,+r removes or adds read-only bit.
-s,+s removes or adds system bit.
dosfile... is one or more dosfiles to operate on.

Note that for all m commands in the mtools toolkit you can specify pathnames with either the UNIX-like forward slash (/) or the DOS backslash (\). Note also that you must quote the \ if you don't want the shell to interpret it as an escape for the following character. For example:

```
$ mattrib +a file.dos
$ mattrib -s "a:\subdir\file.dos"
```

See the following commands for a discussion of a: and how you can manipulate this.

med. The syntax is:

```
mcd [dir]
```

where dir is directory to change to. Without a specified dir, you will get the current working directory displayed.

mcopy. The syntax is:

```
mcopy [-mntv] file1 file2
```

or

```
mcopy [-mntv] file1 ... dir
```

where -m preserves file modification time.
-n—no warning displayed when overwriting an existing file.
-t copies as a text file (converts CR/LF to LF or vice versa).
-v is verbose mode.
file1, file2 specifies files to be copied.
dir specifies a directory to copy files to.

This command copies files and performs the proper file modifications if specified.

mdel. The syntax is:

```
mdel [-v] file...
```

where -v is verbose mode.
file... is one or more files to delete.

The command deletes DOS files.

mdir. The syntax is:

```
mdir [-w] dir
```

where -w is wide output.
dir is directory to be displayed.

mdir displays the contents of a DOS directory.

mformat. The syntax is

```
mformat [-t tracks] [-h heads] [-s sectors] [-l label] drive:
```

where -t tracks specifies the number of tracks on the device.
-h heads specifies the number of sides on the device.
-s sectors specifies the number of sectors on the device.
-l label writes a label on the device.
device: is device to be operated on.

One thing to note with mformat is that you must preformat the disk with the UNIX format command. mformat will place DOS information such as boot sector, FAT, and root directories on a preformatted UNIX

disk. Once mformat has been run, you can use this device on any DOS machine.

mkmanifest. The syntax is:

```
mkmanifest [files]
```

where files are long-named files to be shortened.

mkmanifest provides you with a listing of the shortened names of long-named UNIX files that may be written out with a command like mwrite. When you write files from a UNIX filesystem to a DOS filesystem, the filenames may need to be shortened to support DOS filename conventions (8+3). If this occurs, you will want to use mkmanifest as follows:

```
mkmanifest file1 file2 ... > manifest.out
```

which will place the shortened names of file1, file2, etc., in the file manifest.out. You can then copy this to the DOS output device for later execution as a script. The manifest.out file will look something like:

```
mv short1 long1
mv short2 long2
```

where shortn and longn are the converted filenames. In other words, this creates a script which you can use later to move the files back from a DOS filesystem to a UNIX filesystem and preserve their original long names.

mlabel. The syntax is:

```
mlabel [-v] drive:
```

where -v is verbose mode.
drive: is drive to operate on.

mlabel generates a label on a DOS output device. You are prompted for a new label.

mmd. The syntax is:

```
mmd [-v] dir ...
```

where -v is verbose mode.
dir... is one or more directories to create.

mmd stands for make MS-DOS directory, and it will create MS-DOS directories on a DOS output device.

mrd. The syntax is:

```
mrd [-v] dir ...
```

where -v is verbose mode.
dir is one or more directories to remove.

mrd stands for MS-DOS remove directory.

mread. The syntax is:

```
mread [-mnt] dosfile unixfile
```

or

```
mread [-mnt] dosfile1 [...] unixdir
```

where -m preserves file modification times.
-n doesn't display a warning when overwriting a file.
-t transfers as a text file (change CR/LF for LF and vice versa as necessary).
dosfile is DOS file to read.
unixfile is UNIX file to write.
unixdir is place to copy one or more DOS files to.

mren. The syntax is:

```
mren [-v] file1 file2
```

where -v is verbose mode.
file1 is file to rename.
file2 is name of renamed file.

mren renames a file from file1 to file2.

mtype. The syntax is:

```
mtype [-st] file [...]
```

where -s strips high bit.
-t is text line viewing.
file ... is one or more files to operate on.

mtype displays the file(s) on standard output.

These are the basic commands included in mtools. More information is available for each command in the specific man page included in the mtools distribution in the main directory with a suffix of .1. See also the Release.notes file for more information on current release information. This is important since new features provide significantly enhanced functions.

8.10.3 Installation

There is a standard makefile included with the mtools distribution which needs some modification based on your machine architecture. Much of this is documented in the file Configure. See this file for more information if this section is not sufficient to build mtools correctly.

Once you select the correct values for your machine, type the command:

```
$ make
```

This should be sufficient to build mtools for your platform.

For example, on the RS/6000 the best choice seems to be the RT_ACIS which defines a floppy drive at /dev/rfd0 addressable as A:. However, the first default floppy device sector information is 15 while on the RS/6000 it should be 18. Make this change and rebuild with the make command. Note also that gcc seems to work better with this than does CC, so you might use a command like:

```
$ make CC=gcc
```

This is useful and seems to work correctly on the RS/6000 used to build this CD's software. Note that you also need to create the CFLAGS with the definitions:

```
CFLAGS=-O -DRT_ACIS -DLOCKF
```

This will create the correct system on the RS/6000. To install mtools, you must ensure that the Berkeley install command is first in your path. This is in /usr/ucb. The default makefile also seems to fail on install and is poorly written for modification. You may want to copy the manual pages and binaries to the appropriate directories manually.

Note that if you need to define additional DOS devices to support, you need to edit the devices.c file, which is the devices database. See the Configure file for exact information on the syntax of the devices.c file.

8.10.4 Conclusion

The utility programs included in the mtools distribution have proven invaluable to many users of UNIX and DOS machines. This gives you the ability to transparently transfer information to and from DOS and UNIX machines. The capability to read, write, and manipulate DOS files on a UNIX platform provides significant functionality and eases the transition from DOS to UNIX.

8.11 cpio

8.11.1 Introduction

cpio is one tool that comes with most standard versions of UNIX and is fairly portable between platforms. Because of this fact and the fact that it is less and less used by UNIX users, it was not discussed as a power tool in the native tools section of this book. However, because of its power and the implementation written by GNU, a brief introduction is included in this section.

cpio is an archive utility much like tar and was tradionally used to create archives for transmission on the Internet and other places. It is still used; however, its frequency of use has dropped over the last few years as the use of tar has increased. With the 2.2 version of GNU cpio, you have the ability to manipulate tar files as well as cpio archives with the cpio command.

GNU cpio has documented advantages over most vendor's implementations of cpio, including network tape drive and symbolic link support.

cpio is a GNU product and as such is subject to its GNU General Public License as included both in the product distribution and in App. C of this book.

8.11.2 Usage

The basic syntax for cpio is:

```
cpio {-o|--create} [-0ABLVacv] [-C num] [-H format] [-M string]
[-O [[user@host:] archive] < list [>archive]

cpio {-i |--extract} [-BSVbcdfmnrtsuv] [-C num] [-E file]
[-H format] [-M string] [-R [user][:.][group]]
[-I [[user@]host:]archive] [-F [[user[@host:]archive]
[pattern...] [< archive]

cpio {-p|--pass-through} [-0LVadlmuv] [-R [user][:.][group]] dir <
list
```

where -0 reads list of files terminated by a null instead of the standard newline character.

-A appends to existing archive.
-B sets blocksize to 5120 bytes instead of the standard 512 bytes.
-C num sets blocksize to num bytes.
-E file reads from file the names of more files to be used as input to cpio.
-F user@host:archive specifies an archive name to use as input and output. You can specify a remote hostname and device as show above. Protection is done with .rhosts (rsh command).
-H format uses format for archive from list below:
bin—obsolete binary format
crc—SVR4 portable format with checksum
newc—SVR4 portable format
odc—old POSIX portable format
tar—old tar format
ustart—new POSIX.1 tar format
-I user@host:archive specifies an archive name to use as input. You can specify a remote hostname and device as shown above. Protection is done with .rhosts (rsh command).
-L copies actual files pointed to by symbolic links instead of link itself.
-M string displays string at end of volume requesting additional volume.
-O user@host:archive specifies an archive name to use as output. You can specify a remote hostname and device as shown above. Protection is done with .rhosts (rsh command).
-R user:.group sets the created files' ownership to user and group.
-S swaps the halfwords of each word (see -b).
-V displays a dot as each file is processed.
-a preserves original access times.
-b swaps order of bytes and halfwords (converts big endian to little endian).
-c uses old archive format (for backward compatibility).
-d makes appropriate directories as needed.
-f copies only those files, *not* machine-specified patterns.
-i,--extract is copy-in mode.
-l maintains links intead of using actual files.
-m preserves file modification times.
-n displays uid and gid instead of username.
-o, --create is copy-out mode.
-p,--pass-through is copy-pass mode.
-r interactively renames files with prompting.

-s swaps bytes with a halfword.
-t is table of contents.
-u replaces all files disregarding date information.
-v is verbose mode.
archive is archive to manipulate.
dir is output directory.
list is list of filenames (can be used similarly to patterns).
patterns is regular expressions to determine files matched.

There are three basic modes of operation for cpio: copy-out(-o), copy-in(-i), and copy-pass(-p). Copy-out mode transfers file input one line at a time into an archive, copy-in mode transfers file from an archive to files, and copy-pass transfers files from one directory structure to another. These three modes provide all access to cpio and its associated archives. Actually, cpio is one of the easiest archive commands of any available on UNIX. The most common way for cpio to be used is with the find command. A simple example of cpio archive creation is:

```
$ find . -name "file*" -print | cpio -o > /dev/rmt0
81 blocks
```

This command will search starting in the current directory for any files starting with the string file and move them to a tape device /dev/rmt0. You could just as easily move them to an archive file with a command like:

```
$ find . -name "file*" -print | cpio -o > ../file.cpio
```

This will create a file named file.cpio in the directory above the current working directory. Note that you want to be careful if you use the same directory to create the archive since you may get a copy of the archive inside the archive itself. *Be careful.*

Some other simple examples of this command are:

```
$ cpio -itv files.cpio
```

This will generate a table of contents listing to your screen.

To move an entire directory hierarchy, use a command group like:

```
$ find . -print | cpio -p /newrootdir
```

This will move the entire directory structure (including subdirectories and their contents) beginning with your current working directory into /newrootdir and will preserve the current structure. This is a very use-

ful command and is often simpler than the tar command that performs the equivalent functionality.

With respect to tar support, cpio will fully support most vendors' tar archives. For example, if you have created a file named kevin.tar and you want to see a table of contents, you could use a command like:

```
$ cpio -itv -H tar < kevin.tar
```

This will generate a standard listing just as the tar command would. You can create and extract from tar files with exactly the same syntax.

As you can see, there are many ways to use the cpio command, all of which are fairly easy. Keep in mind that there are various formats, and you may have to use a format other than the default for portability purposes. However, if you use GNU cpio, it will support all formats, and you shouldn't have to worry about portability.

8.11.3 Installation

The cpio installation is very straightforward and consists of the standard GNU configure subsystem. This means you can simply run the commands:

```
$ configure --prefix=/usr/local
$ make
$ make install
```

This will generate three exectables: cpio, mt, and rmt. The mt and rmt commands are included to support cpio's operation on your machine. If you want to use a remote tape device, the rmt command will support this function if one is not available on your machine. The mt command is used to manipulate a local tape device if this is necessary. The usage of this should be transparent to you; however, you can simply type:

```
$ mt
```

to get a listing of available options for the mt command. This does not work for rmt since this is merely an interface to mt on another system.

If you are building this product on an RS/6000, you want to issue the command:

```
$ make CFLAGS=
```

unless you have the PFT U211273 installed since you will get a compiler error otherwise. It also seemed to work well with gcc, so you might try this as well as your native compiler.

8.11.4 Conclusion

cpio from GNU has advantages over the vendor-supplied cpio, including remote device and tar file support. These features as well as the increased portability of having one cpio on all vendor platforms makes GNU cpio a good choice if you are using archives in a heterogeneous environment.

See the man pages for more information on both cpio and mt.

8.12 ispell

8.12.1 Introduction

ispell is an interactive spell checker which presents its best guesses as to the correct spelling of a misspelled word. You can either use it in an integrated fashion with emacs or by itself. If you have need for a spell checker to check ASCII files for typos and misspelled words, ispell is probably the best way to go.

ispell is a GNU product and as such is subject to its GNU General Public License as included both in the product distribution and in App. C of this book.

8.12.2 Usage

The basic syntax for ispell is:

```
ispell [-l | -D | -E] [file...]
```

where -l displays misspelled words from standard input.
-D displays system dictionary flags.
-E displays system dictionary expanded.
file ... is one or more files to spell check. If no file is entered, ispell uses standard input.

The most common way to use ispell is on a flat ASCII file. Once you enter the ispell command, it begins to scan the file until it finds a word not in its dictionary. At this time it places the unknown word at the top of the screen and prints a series of "near miss" words below it. At the bottom of the screen, the two lines surrounding and including the line with the unknown word are displayed.

Once the words are displayed, you can select from the following options:

?	Displays help
SPACE	Accepts the word this time
A	Accepts the word for the rest of the file
I	Accepts the word and places it in the user's private dictionary

R	Prompts for a replacement word
num	Where num is the number is the near miss that is correct

You can create your own private dictionary $HOME/ispell.words as well as maintain a system-level dictionary /usr/lib/ispell.dict.

The easiest way to add words to your dictionary is to use the commands listed above. You can also add words using a procedure described in the README file. See this for more details.

This distribution also contains a texinfo file which has extensive documentation on ispell. You should definately investigate this file and read it before beginning to use ispell. If you are unsure as to how to use the texinfo file, see Sec. 8.3 for more details.

8.12.3 Installation

Because ispell is a GNU product, it comes with the standard configure capability which all users have come to expect from GNU. The methodology used to build ispell is no different from that described for other GNU products in this book:

```
$ ./configure --prefix=/usr/local
$ make clean
$ make
$ make install
```

This will build ispell and place the bin and man pages in the appropriate directory. The other file of interest is the file ispell.dict. This is the dictionary file which is created by the build procedure. This is placed in a lib directory, and ispell is compiled with a hard-coded path for this file. You will see the path on the first execution of ispell. In the above example, ispell will expect ispell.dict to be in /usr/local/ispell/lib. If it is not, ispell will fail. Keep this in mind as you build ispell.

The other aspect to this is that you may want to build your own dictionaries. This is beyond the scope of this section; however, it is documented in the INSTALL file contained with the distribution. See the end of this document for assistance with developing and building your own dictionaries.

8.12.4 Conclusion

While ispell is a relatively simple tool to understand and use, it does provide value when you need to check the accuracy of your typing or spelling. If you can extract information into a flat ASCII file, you can use ispell very effectively. You can also use the integrated ispell with emacs. See Chap. 9 for more information on this capability.

8.13 monitor

8.13.1 Introduction

monitor is a real-time system monitor for AIX. You can display a variety of statistics including disk I/O, CPU utilization, memory, and paging and swapping statistics. monitor will also display statistics on individual processes, including memory utilization, CPU utilization, and many other useful metrics.

8.13.2 Usage

The basic syntax for monitor is:

```
monitor [-s time] [-ru] [-top [number]] [-all] [-disk] [-net]
```

where -s time sets the screen update time for measurements.
-r displays running processes.
-u displays the userid of the users.
-top number displays the top CPU processes and a summary of system variables.
-all displays all system and top CPU process events.
-disk displays detailed disk statistics.
-net displays detailed network statistics.

An example output display looks like:

```
AIX monitor v1.12: Snoopy        Wed Jun 15 18:48:04 1995
Sys 5.3% Wait 0.0% User 94.7% Idle 0.0%    Refresh: 10.56 s
0%          25%         50%         75%         100%
===>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
Runnable processes 8.33 load average: 8.85, 9.01, 8.76

Memory      Real        Virtual     Paging (4kB)          Process events        File/TTY-IO
free        85.6 MB     431.6 MB    1.6 pgfaults          189 pswitch           51 iget
procs       84.7 MB     168.4 MB    0.0 pgin              338 syscall           24 namei
files       21.7 MB                 0.0 pgout             32 read               0 dirblk
total       192.0 MB    600.0 MB    0.0 pgsin             23 write              42237 readch
                                    0.0 pgsout            0 fork                30173 writech
DiskIO      Total Summary                                 0 exec                1 ttyrawch
read        0.0 kByte/s                                   2 rcvint              0 ttycanch
write       0.0 kByte/s
            Client Server NFS/s   39 xmtint  882 ttyoutch
transfers   0.0 tps    0.0       0.9 calls
active      0/11 disks            0.0        0.0 retry
                                  0.0        0.0 getattr          Netw read  write
TOPdisk read write    busy        0.0        0.9 lookup lo0       0.0 0.0 kB/s
hdisk1      0         0 kB/s      0%         0.0        0.0 read
en0         3.8       34.3 kB/s
hdisk2      0         0 kB/s      0%         0.0        0.0 write
hdisk3      0         0 kB/s      0%         0.0        0.0 other
hdisk4      0         0 kB/s      0%

Snoopy                load averages: 1.05, 1.10, 1.14          Sun May 15 18:41:25 1995
Cpu states:           93.1% user,          5.0% system,        2.0% wait, 0.0% idle
Real memory:          110.9M free          68.4M user 12.6M numperm       192.0M total
Virtual memory:       466.6M free          133.4M used         600.0M total
```

```
PID USER         PRI     NICE   SIZE     RES STAT      TIME       CPU%      COMMAND
53453 mheinila   105     0      482K     616K run      95:26      94.1%     prg1
514 root         127     21     28K      8K run        8157:10    2.0%      wait (kproc)
96310 jmaki      61      0      299K     332K run      0:00       2.0%      monitor
16397 amiettin   60      0      258K     88K sleep     0:03       1.0%      ybiff
55547 jhi        60      0      152K     52K sleep     0:00       1.0%      rlogin
12353 root       60      0      885K     220K sleep    349:49     0.0%      glbd
11836 root       60      0      504K     172K sleep    306:59     0.0%      llbd
771 root         39      21     32K      16K sleep     201:37     0.0%      netw (kproc)
106601 hsirvio   60      0      1348K    12K sleep     181:10     0.0%      sas.exe
0 root           16      21     24K      8K sleep      111:51     0.0%      swapper (kproc)
5109 root        60      0      156K     128K sleep    103:51     0.0%      portmap
1 root           60      0      280K     216K sleep    99:04      0.0%      init
2650 root        60      0      64K      28K sleep     75:46      0.0%      syncd
59556 root       60      0      656K     356K sleep    47:20      0.0%      mumsm
9509 root        60      0      146K     12K sleep     43:43      0.0%      nfsd
8995 root        60      0      138K     12K sleep     43:40      0.0%      nf
```

As you can see, monitor is quite comprehensive in its output of information. This is a good tool for developers and system administrators to use to monitor and tune the performance of a system. This will also give you some feeling about the efficiency and operation of your software systems under AIX.

8.13.3 Installation

The installation of monitor is very straightforward. First examine the makefile for any options that you may wish to change such as executable optimization, compiler you wish to use (native or gcc), etc., and save the changes. One other thing you may want to change is INSTALLDIR to ensure that you place the resulting files in the appropriate directories.

Next, issue the command:

```
$ make
```

This will build the executable monitor. You can install the executable in the default location with the command:

```
$ make install
```

This will install all resulting files in /usr/local.

8.13.4 Conclusion

monitor is a very useful and comprehensive tool to use when examining the current performance of a machine. By using monitor to study the current load characteristics, you can more clearly understand the impact certain software systems are having on the overall performance characteristics of the machine.

8.14 sysinfo

8.14.1 Introduction

sysinfo is a tool which provides a vast amount of information about the particular system it is running on; information such as hostname, network addresses, CPU model and type, kernel achitecture, operating system name and version; and information about the devices connected to the system. It is a very useful tool for software systems to take advantage of in installation and configuration scripts since you can get much of the information you need to build a software system appropriately for any given machine type.

8.14.2 Usage

The basic syntax for sysinfo is:

```
sysinfo [+|-all] [-debug][-level level1,level2,...]
[-offset amount] [-show item1,item2,...] [+|-terse]
[+|-unknown][+|-useprom] sysinfo -list [level|show]
sysinfo -version
```

where +|-all enables (+) or disables (-) all known information.
-debug enables debug messages.
-level levels sets the show level (see -list).
-list level|show lists all possible values for both level and show.
-offset amount sets the number of spaces to offset when printing device information.
-show items displays information about only those items in items.
+|-terse enables (+) or disables (-) terse format.
+|-unknown enables (+) or disables (-) those devices that appear on the system but are unknown to sysinfo.
+|-useprom enables (+) or disables (-) using values obtained from the system PROM.
-version displays the syinfo version.

sysinfo works on a variety of systems including AIX. The only issue directly relating to the RS/6000 in the accompanying documentation is the fact that you must ensure that the environmental variable LANG is set correctly so that sysinfo can find all the appropriate device information and give you accurate information.

8.14.3 Installation

To make sysinfo on an RS/6000, you must modify the makefile. Fortunately, there is one included that is already modified for the RS/6000

running AIX. To invoke the correct build sequence, issue the command sequence:

```
$ make clean
$ make -f Makefile.rs6k
```

This will tell make to use the makefile Makefile.rs6k instead of its default. This will build the sysinfo executable. You can install this with the command:

```
$ make -f Makefile.rs6k install
```

if you want to install the executables and man pages in the default /usr/local directories. This is all there is to building sysinfo.

8.14.4 Conclusion

sysinfo is a very useful tool which provides vital system information regarding the current configuration of a variety of types of UNIX machines. This can be very useful to include with software distributions to ensure that the makefiles are structured correctly. This is especially useful when a software system is sent to a variety of different types of UNIX machines. Use sysinfo to establish the machine type and help your installation and configuration scripts do the rest.

8.15 xzap

8.15.1 Introduction

xzap is a graphically based process killer. It is based on the zap program documented in the Kernighan and Pike book *The Unix Programming Environment.* It provides a simple way to kill processes from a GUI instead of requiring the more cumbersome command line interface commands.

8.15.2 Usage

The xzap command creates a window on the screen from which you can send signals to processes. The basic xzap screen is shown in Fig. 8.4. The basic syntax for xzap is:

```
xzap [-toolkitoption...][ps options]
```

where -toolkitoption... consists of:
command
defaultSignal

pidColumn
psOnDeiconfy
scrollHeight
rootCursor
rootTitleSuffix
rootIconNameSuffix
signalList
zapAccelerator

ps option consists of:

quit
zap
help
clear
ps

The usage of the above options can either be from the command line or from within resource files. A resource file is an X11 file which determines how applications react given certain situations. While it is not within the scope of this book to discuss this further, suffice it to say that all of the above resources and more (all documented in the xzap man page) can be input from either the command line or in resource files to further refine and control xzap's behavior. See the man page for more details.

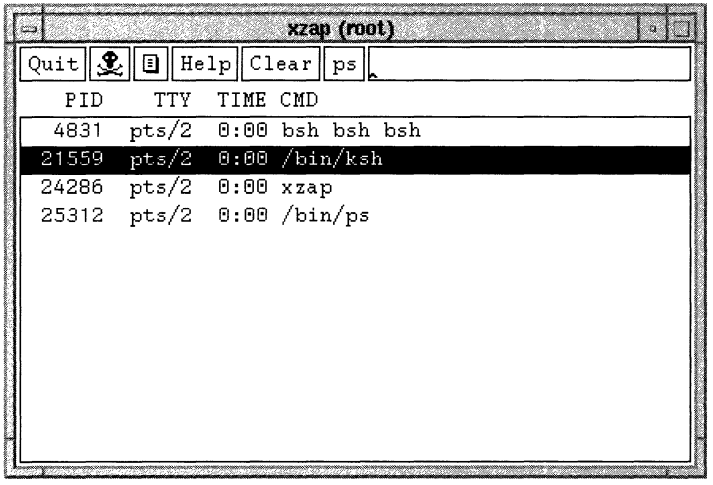

Figure 8.4 The xzap screen.

8.15.3 Installation

It is relatively easy to build xzap on AIX 3.2.4 or later. There is a makefile which has been created for you. If you are running AIX 3.2.4 or greater, you can simply type:

```
$ make -f Makefile.rs6k clean
$ make -f Makefile.rs6k
```

Then to install the images, you can use the command:

```
$ make -f Makefile.rs6k install
```

This will copy the generated files to the appropriate directories in /usr/local. If you want to change the locations of the output files, you will need to change the makefile to reflect the appropriate files.

If you are running AIX 3.2.3, you will need to follow the instructions in README.rs6k to first build a new makefile and then execute the resultant makefile to generate xzap.

8.15.4 Conclusion

xzap provides an easy way to kill processes without requiring use of the command line. This is a very useful tool for killing runaway processes or processes created by a software system in debug mode.

Chapter

9

The GNU emacs Editor

There are very few editors available from the Internet which warrant discussion in this book. The one exception is GNU emacs. emacs is simply the most powerful editor available today and provides complete extensibility and portability across many platforms and architectures.

Editors are like religions in nature and become most people's primary interface to an operating system along with a command interpreter. While there are other editors available today such as elvis, which is a vi/ed rewrite from the GNU people, by far the most pervasive nonnative editor is emacs. The most important thing about an editor is its portability and availability. While these other editors exist, they are most likely not on any platform you have and therefore won't be very useful when moving onto a new machine. Because of this, this book limits itself to a discussion of emacs.

9.1 Introduction

emacs is the premier editor avilable in the UNIX market today. It has capabilities far beyond most free editors and, in fact, beyond most editors that cost significant amounts of money. It was written by Richard Stallman of GNU fame and continues to be one of the best and most widely used editors available on the UNIX platform today. In fact, ports of emacs have been made to Macs, PCs, VAX/VMS, and many other platforms. It is clearly the most widely ported editor available for almost any platform today.

Because of the popularity of emacs, it has been widely ported and is supported and distributed by several companies. emacs is written in LISP, and because of its complexity, some companies have made enhancements and charge a fee for emacs. Companies like Unipress Soft-

ware and CCA provide emacs for a fee, while there is still a free version of emacs called GNU emacs. This is the version that will be discussed in this chapter.

emacs stands for "editing macros" and was originally designed as a set of editing macros for an ancient editor known as TECO. This editor has since become obsolete, but emacs has continued to grow and thrive. This is a tribute to its power and flexibility. It is somewhat misleading to call emacs an editor. In fact, emacs is an entire working environment and has integrated mail and shell capabilities and a knowledge of your environment which allows it to help you in programming and compiling. emacs is an extremely sophisticated programming environment and, because of its sophistication, is a favorite of software developers and power users.

GNU emacs, as you would expect, is developed and distributed by the GNU Free Software Foundation (see Chap. 6 for more information on this organization). This means that it is freely available and of very high quality. In fact, most people consider the GNU emacs implementation the best of all emacs versions including those sold by other organizations.

The version of emacs discussed in this chapter is 19.28. Because of the constantly changing nature and complexity of emacs, this chapter will focus on discussing the main capabilities and power of emacs and leave the complexities and advanced capabilities of emacs for another book. The purpose of this chapter is to get you familiar with emacs and to provide basic capabilities and knowledge of the product.

9.2 Installation

With version 19 of emacs, the installation has been greatly simplified and the functionality of the install procedure has been enhanced. Through the use of configure, you can select many options related to the compilation and installation of emacs on a single command line.

There are many files in the emacs distribution, several of which are key to the emacs installation process. This chapter will discuss each in turn as they are needed for the installation process.

The first file to examine in the INSTALL file. It documents much of what you need to know to configure and build emacs. The first thing it mentions is to ensure that you have enough swap space on your machine since it requires approximately 8MB or more of swap. If you get an error from the temacs command or when executing the dumped emacs, increase your swap space and try the build again. Note that temacs uses a file called lisp/paths.el which references several files related to other nonemacs utilities. You may encounter problems related to these, so be careful.

Once you have examined the INSTALL file, you will need to carefully

examine the etc/MACHINES file for details on your particular UNIX implementation and the issues related to the building of emacs on your particular platform. Read your specific architecture section carefully since there are often very specific notes about installation problems and issues. For example, on a Sun SPARC machine, the file specifies that you need to be explicit in your choice of platforms. This is key to the success of building the product.

The first command you will issue is the configure command. The basic syntax is:

```
configure architecture [options...]
```

where architecture is the architecture of your machine chosen from the etc/MACHINES file.

options consists of one or more of the following:

--exec-prefix=EXECDIR—specifies a directory where the architecture-dependent executable files will be placed (EXECDIR/bin). The nonarchitecture dependent files such as source code will be left in their normal directories. Other architecture-dependent files will be placed in EXECDIR/lib/emacs/VERSION/CONFIGURATION.

--prefix=DIR—specifies the directories in which to place generated emacs files.

--run-in-place—specifies that you would like to maintain the emacs directory structure just as in the distribution and not install it in the /usr/local default directories.

--srcdir=DIR—specifies the emacs source directory.

--with-gcc—uses the GNU C compiler.

--with-x11, with-x=no—specifies whether you would like the X11 interface or not.

--x-includes=DIR—specifies the X11 include file directories.

--x-libraries=DIR—specifies the directories for the X11 library files.

The configure command is documented in more detail in Sec. 7.4. See this for more information if you have trouble with the emacs installation. The first section in the INSTALL file goes into more detail on this, so reference it if you need more information.

The best way to learn to use configure for emacs is to try it a few times and see what happens. Note that when you run configure, you do not compile or build anything but instead create the appropriate makefile and associated build files, which you can then execute to build the product.

A simple example of running configure on an RS/6000 machine running AIX is:

```
$ configure rs6000-ibm-aix3.2.5 --with-x-11
--prefix=/usr/local/emacs/emacs-19.28 --run-in-place
```

This will generate the appropriate makefile from which you can build emacs. Issues such as compilers, libraries, utility commands, and many other details are handled automatically so that the makefile will execute correctly.

The resulting file from the configure command is config.status. Look at this if you want to see exactly what happened during the configure process.

There are a variety of files including ./list/paths.el, which contains information which tells emacs where to find executables like news readers, sendmail, and others which you can use in conjunction with emacs. A corresponding file, ./lisp/site-init.el, is the file you need to edit if you want to change the configuration of the paths.el file. You should *not* edit the paths.el file directly due to the nature of the complex syntax. You may need to edit the ./listp/site-init.el file depending on the configuration of your machine. Use the seq command in site-init.el instead of defvar as used in ./lisp/paths.el as described in the ./list/site-init.el file.

Once you have configured the new makefile, you can execute the make command. There are several options to the make command which will give you quite a bit of control over the build process. The basic syntax for the make is:

```
make [install] [option=value, option=value, ...]
```

where install is optional and tells the make to place all resultant executables, libraries, man pages, and other special files in the default location, which is /usr/local/bin, /usr/local/lib/emacs/VERSION/lisp, etc.

options consist of:

- bindir—location of binary files
- datadir—architecture-independent emacs data files directory
- libdir—emacs library directory
- prefix—as described for configure, the directory which is the root for all emacs files instead of the default /usr/local
- statedir—architecture-independent shared emacs files directory

There are several other variables that are emacs specific. These variables are documented in the INSTALL file if you want more information. Because you have run configure, you can simply invoke:

```
$ make
```

to build emacs.

The other files that will be of interest are contained in several different directories. The first is the PROBLEMS file. This lists many of the known problems with emacs and documents either a work around or a fix. View this before you begin to use emacs since it will save you time as you proceed.

In fact, there are README files in all directories as well as ChangeLog files in most. The ChangeLog files document the changes in the files in each particular subdirectory from version to version. These are important to view if you are interested in seeing what changes have taken place in the new version.

Finally, there are a variety of files in the etc subdirectory which may be of interest to you. Files ranging from specific information on a particular kind of computer to cookie recipes to man pages. Note that the man pages end, as is the norm, with a .1 suffix. Use nroff on the man page for more information. Probably the most interesting file in the etc subdirectory is the FAQ file. This contains many of the most commonly asked questions about emacs and is extremely useful for the new emacs user to read before beginning to use emacs. See this file before you read the rest of this chapter.

One of the keys for the generation of the executables is to ensure that you have installed the development files for X11. This means that all of the appropriate include and library files are in the proper X11 directories. If you have not done this, you will see error messages saying that the build cannot find a file with a .h extension or there are unresolved symbol references. This probably means that you haven't installed all the appropriate files for the X11 portion of emacs to compile correctly.

If you are interested in service and assistance with the emacs product, see the file etc/SERVICE for a listing of consultants and others who will provide maintenance and other support for emacs and a variety of other free software products. This list is duplicated in App. D of this book.

9.3 Usage

The basic syntax is:

```
emacs [-q] [file]
```

where -q begins emacs without reading any initialization files (.emacs). If you invoke emacs without a file, you are automatically dropped into the help screen. From here you can invoke any emacs command.

An initialization file named $HOME/.emacs is read upon invocation.

emacs is primarily a command-driven editor which responds to keystrokes in some combination with the CTRL character. These keystrokes execute a corresponding LISP function. Because of the long and cryptic name of commands, you will want to learn the keystrokes and forget about the command names.

There is both a built-in tutorial for emacs and a file named etc/TUTORIAL which you should reference if you are interested in learning more about emacs.

9.4 The emacs Screen

emacs is a full-screen editor and supports many different types of terminals and keyboards. The standard of screen is shown in Fig. 9.1. The main area of the screen is the input area, which is where all typed text will be placed. This is a full-screen area, and you can move the cursor freely from one area to another. The line second to the bottom is the mode line. It contains ** if you have made changes to the file since it

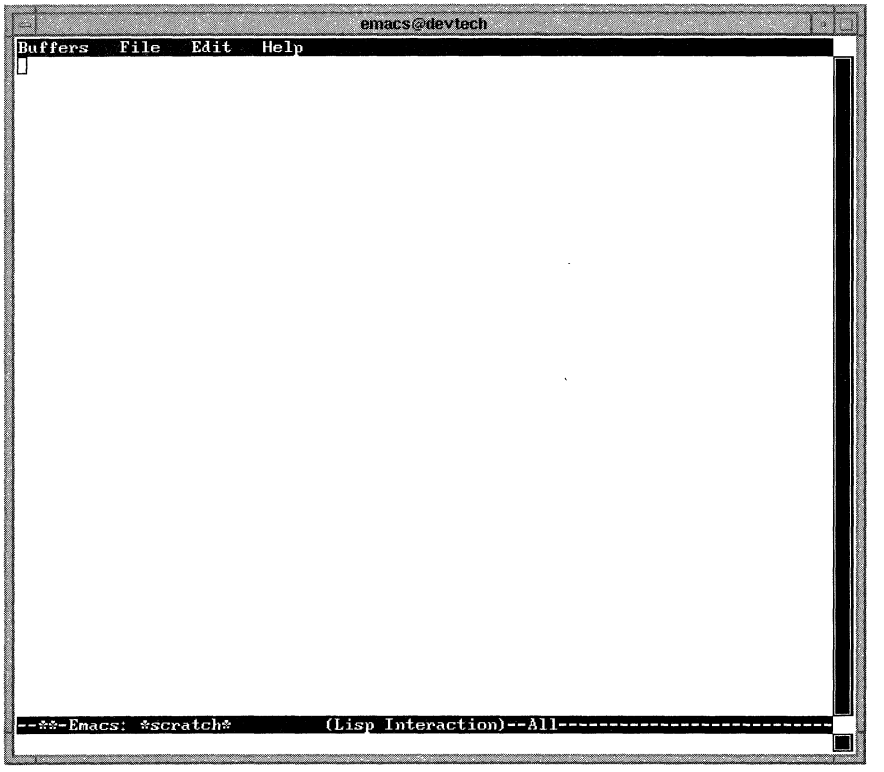

Figure 9.1 Standard emacs screen.

was last changed. To the right of this is the string Emacs:. Following this is the name of the buffer (which may or may not match the name of the file). In parentheses is the name of the mode (see Sec. 9.5) which shows you which mode you are in. Finally, your position in the file is displayed as a percentage from the first to last line. If the entire file is displayed, the string All is displayed. The last line on the screen is the minibuffer. It displays all commands and filenames that you enter.

9.5 emacs Modes

emacs has a series of modes which dictate how it behaves when certain keystrokes are executed. Modes give emacs knowledge of your particular environment and give you enhanced capabilities that will make you more productive. A basic listing of modes is:

Mode	Use
C	When writing C code
emacs LISP	When writing emacs LISP
Fortran	When writing Fortran
Fundamental	Default mode; no special behavior
Indented text	Indents all text
LaTeX	formatting LaTex files
LISP	Writing LISP programs
LISP interaction	Writing and executing LISP
Outline	Writing outlines
Picture	Creating simple drawings
Scribe	Formatting Scribe files
Text	Writing text
TeX	Formatting TeX files
View	Viewing files in read-only mode
nroff	Formatting nroff files

Modes are determined by emacs on invocation. emacs examines the file extension and determines which mode to enter. If it connot determine mode from the file entension, it examines the contents of the file. If it cannot determine the mode, it places you in the default mode Fundamental. This mode assumes no special characteristics.

To move from one mode to another, simply type:

```
ESC x modename RETURN
```

where modename is one of the above mode names. This allows you to jump back and forth between modes easily.

There are several minor modes which determine certain default be-

haviors inside a particular major mode. Minor modes are things like Abbrev, Fill, Overwrite, and Autosave mode. These provide default characteristics within a major mode.

9.6 emacs Commands

The commands in emacs are primarily written in LISP and consist of powerful but cryptic commands which are invoked either by command name or keystroke. The keystrokes are by far the most common way to invoke emacs commands because of the cryptic nature of the command names.

The keystrokes for emacs commonly begin with an ESC or CTRL. The basic keystroke operations look something like:

`C-x`	Where x is any character.
`C-c string`	Where string consists of a string or additional control sequence; generally related to modes.
`C-x string`	Where string consists of a string or additional control sequence; generally file-related commands use C-x string.
`ESC x`	Where x is any character.
`ESC x string`	Where string is any emacs command name. This is the general-purpose way to invoke an emacs command.

Note that the notation C-x means that the CTRL key must be held down while the appropriate key is pressed. Both are then simultaneously released. This method of depressing two keys simultaneously is fundamental to the operation of emacs. While this leads to the need to use two hands to type and use emacs effectively, it also removes most of the confusion and inconsistency of keyboard mapping and differences. This is particularly important in heterogeneous environments with multiple operating environments and keyboard layouts. emacs is certainly the most portable multiplatform editor available today. Don't let the method of multikey operation deter you from using emacs and appreciating its full power and potential. For the ESC key, on the other hand, you simply press it and then the appropriate key. It is not necessary to continue to hold down the ESC key when depressing the other key. This is fundamental to the operation of the keyboard, and you will rapidly become used to this mode of operation. Your fingers will just learn where to go.

9.7 Help and the Tutorial

The most important thing to learn when moving to a new environment, particularly something as interactive as an editor, is where to go for more information and help. Interactive help is one of the most powerful

of the features in emacs. There is a very sophisticated on-line tutorial which will introduce you to the emacs editor. This is probably the best place to start.

To invoke the tutorial, type:

```
C-h t
```

This invokes a tutorial, which leads you through much of the basics of emacs. The screen is shown in Fig. 9.2. Once you are through with the tutorial, you will be much better prepared to work in emacs and, therefore, will be much more productive.

Along with the tutorial, there is a general help facility which provides an enormous amount of documentation on the emacs system. To invoke the help facility, type:

```
C-h command
```

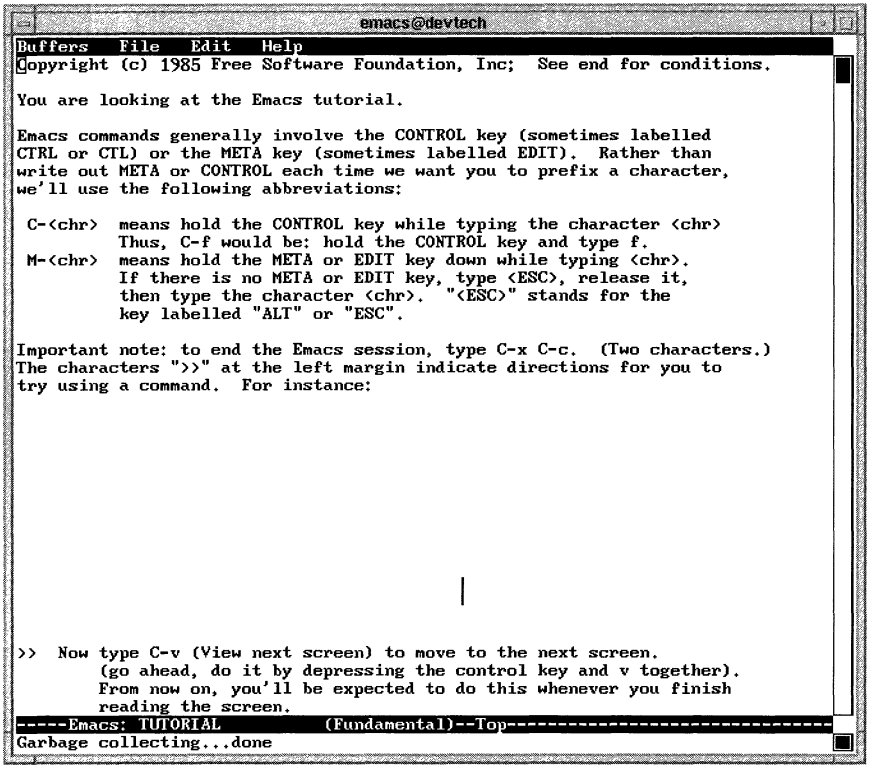

Figure 9.2 Screen for c-h-t.

where command is:

C-h	Displays help on the help facility
b	Lists all key bindings for current buffer
c	Describes command sequence executed by keystroke
f	Displays actions of function
k	Describes command sequence and what happens when keystroke is pressed
l	Lists last 100 characters typed
m	Displays current mode
s	Displays syntax table for current buffer
v	Displays variable value and its definition
w	Displays key binding for command

These commands all prompt for input after execution.

The best way to learn more about emacs help is to type:

```
C-h C-h
```

This places you in a help facility which describes the help facility itself. Once you have gone through this, you will know how to get help in most situations. This is very important because you will probably never know all emacs commands and actions, and the help facility will prove vital in your productive use of emacs.

When you press C-h ?, you get a list of letters and commands across the bottom of the screen, as shown in Fig. 9.3. These provide you with some of the choices described above. To abort the help function, press any other key than one of those listed. Keys such as C-h f will prompt for a function to describe, while others will prompt for a keystroke to be executed. Each is dependent on context and form. See the above descriptions for more information.

9.8 emacs apropos

On some UNIX systems there is a help system known as apropos. Loosely interpreted this means "sounds like." Apropos provides you with a facility which you can search for something when you don't known quite what you're looking for. To execute apropos, type:

```
C-h a
```

This will place you in a mode where you can enter a regular expression, and emacs will search its help database for any matching strings. For example, type:

```
C-h a (reg-exp) compile
```

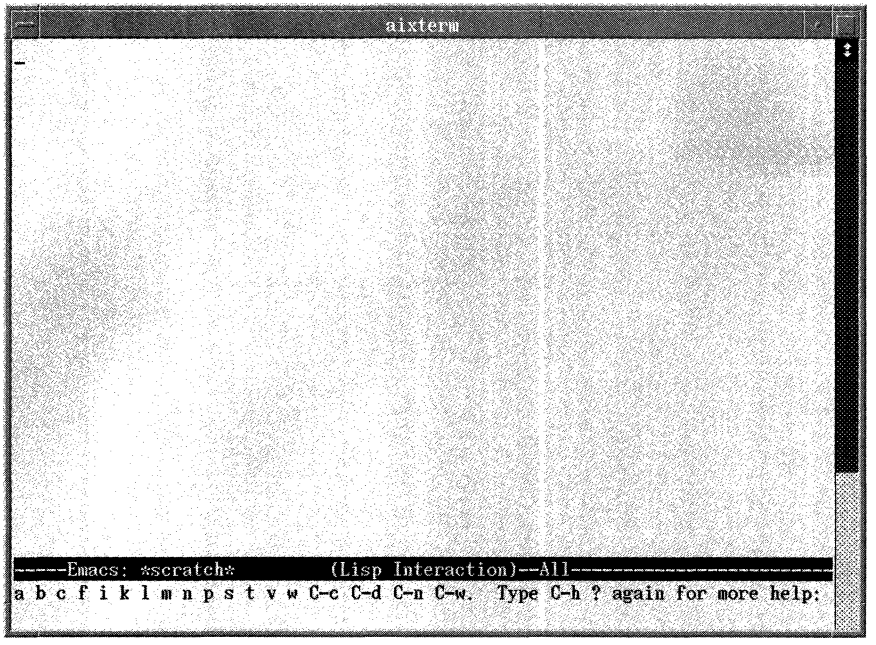

Figure 9.3 The emacs help screen.

This will generate a screen like that shown in Fig. 9.4. Note that it tells you exactly what it has relating to the compile string. Use this when you can't remember how to do things, but you have some idea what you want to do.

There is one other tool within emacs that you should know about. It is called info and really consist of a hypertext database of text containing information about emacs and related tools. To invoke info, type:

```
C-h i
```

Once inside info, you can type h to get an on-line tutorial. Try this to learn more about info. A full discussion of info is beyond the scope of this book; see related emacs documentation for more information on it.

9.9 Getting Current Information

You can get current information on emacs by pressing the command sequence:

```
C-h n
```

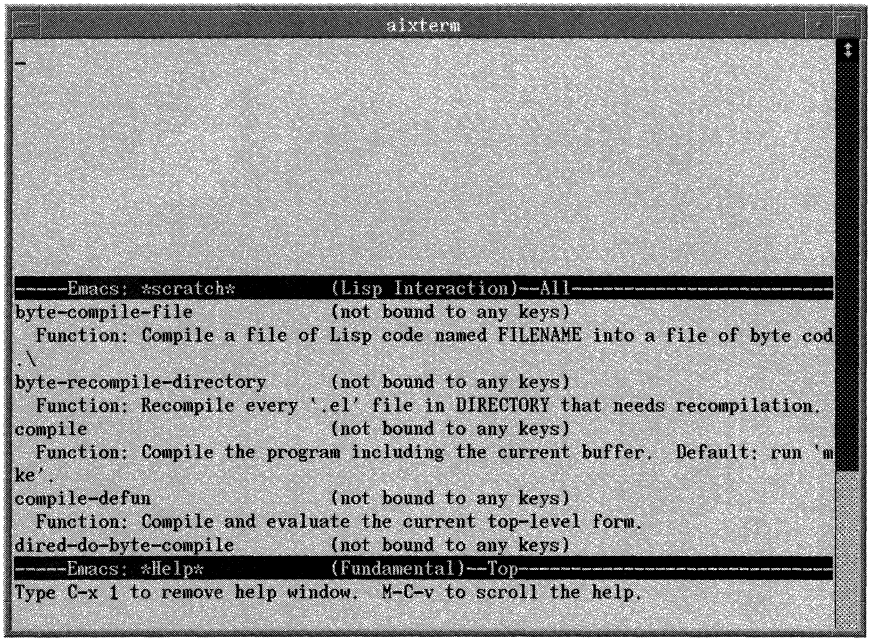

Figure 9.4 Example of regular expression help.

to invoke the news feature of emacs. This will provide "up to the release" information on emacs and associated products. See the C-h minibuffer line for more possible commands.

9.10 Reading in a File

There are two basic ways to read in a file. The simplest way is to simply type the filename on invocation of the emacs editor. For example:

```
$ emacs file
```

Once you are in emacs, you are actually in a buffer named file. If you determine you have read in the wrong file, issue the following command from within emacs:

```
C-x C-f
```

emacs will prompt you for the filename to read in and will issue a command to read the correct file into a new buffer with a matching name. emacs uses the current working directory as a default.

emacs has a nifty feature called command completion. This allows you to type an abbreviated command, which emacs will finish for you.

You simply type the shortest unique string and press TAB, and emacs will finish the string for you. Note that the string must be unique; if it is not, emacs will prompt you with a list of possibilities. You must type enough characters to make it unique and again press TAB. emacs will finish the string for you. This is particularly useful for long filenames. For example:

```
C-x C-f get TAB
```

will expand the file get to any full filename in my current working directory. If the file is named gethostbyname and it is the only filename beginning with a get in my current working directory, emacs will read in the correct file. This can be a real timesaver and is something you will use more and more as you use emacs.

9.11 Making Changes in a File

emacs, unlike many other editors, is in insert mode by default. This means that when you move the cursor to the text input area and begin to type, you insert text and current text is simply moved to accommodate inserted text. If you are more comfortable working in the more normal overstrike mode, type the command:

```
ESC x overwrite-mode
```

emacs works as you would expect a full-screen editor to work. With respect to things like word wrap, you can set the "fill mode" with the command:

```
ESC x auto-fill-mode
```

This will cause word wrapping as you might expect it to work. You can issue the above command again to toggle out of fill mode. If you don't use fill mode, emacs places a backslash at the end of every line as a reminder that the line is continued below.

After using emacs, you will notice that most commands that begin with an ESC operate on words or other groups of characters, while commands preceded by a CTRL operate on single characters. Keep this in mind as you proceed.

There are basic commands to move the cursor within the text window:

`C-f`	Moves forward one space
`C-b`	Moves backward one space
`C-n`	Moves to the next line
`C-p`	Moves to the previous line
`C-a`	Moves to the beginning of the line

`C-e`	Moves to the end of the line
`ESC f`	Moves forward one word
`ESC b`	Moves backward one word
`ESC e`	Moves forward one sentence
`ESC a`	Moves backward one sentence
`ESC ]`	Moves forward one paragraph
`ESC [`	Moves backward one paragraph

Definitions of sentences and paragraphs default to two spaces after a punctuation mark and blank lines, respectively. You can modify these definitions with the sentence-end and paragraph-end variables. See Sec. 9.24 for more details.

There are basic commands to move and manipulate screens:

`C-v`	Moves forward one screen.
`C-l`	Places the current line at the top of the screen and scroll the rest.
`ESC v`	Moves backward one screen.
`ESC >`	Moves to end of file.
`ESC <`	Moves to beginning of file.
`ESC x goto-line n`	n is the line number you wish to go to.
`ESC x goto-char n`	n is the character number in the file you wish to go to.

With emacs, you can also issue a command and tell it to execute it a certain number of times with the command:

```
ESC n command
```

where n is the number of times to execute command. In other words, the ESC n command must precede any emacs command which you wish to execute n times in succession.

There are also simple commands which perform text manipulation shortcuts. For example:

`C-t`	Transposes two letters
`ESC t`	Transposes two words
`C-x C-t`	Transposes two lines
`ESC c`	Capitalizes the first character of the current word
`ESC l`	Puts current word in lowercase
`ESC u`	Puts current word in uppercase

There are also basic text formatting commands that you can execute to justify and center text:

`ESC s`	Centers current line
`ESC x center-paragraph`	Centers current paragraph

9.12 Undoing Commands

You can undo a previous command with the emacs command:

```
ESC x u
```

This will prove invaluable at some point in your emacs usage—don't forget it.

There are also simple ways to move text around in and between buffers. Some of the more common commands are:

C-d	Deletes the character under the cursor.
ESC d	Deletes what remains of the current word.
C-k	Deletes what remains of the line. Note that it takes two C-ks to remove an entire line: the first to remove the text and the second to remove the newline.
DEL	Deletes the character preceding the cursor.
ESC DEL	Deletes previous word.
ESC k	Deletes current sentence.
C-y	Yanks. Restores what you have deleted.
C-w	Deletes marked region.

When you delete things with any of the above commands except the DEL and C-d commands, emacs saves them in a buffer known as a kill ring. You can issue the C-y command to restore all that has been deleted. emacs is smart enough to restore them in the order you deleted them. This means that you can delete two consecutive lines with C-k C-k C-k C-k, move the cursor somewhere else and issue a C-y command. The two lines are restored at the position of the cursor and other lines are adjusted accordingly.

9.13 Working with Text Blocks

One of the most common things you will do is to move blocks of text around in a buffer. The commands to accomplish this are:

C-@or C-SPACE	Marks beginning or end of a region
C-x C-x	Exchanges location of cursor and mark
ESC h	Marks paragraph
C-x C-p	Marks page
C-x h	Marks buffer
C-w	Deletes marked region
ESC h	Marks current paragraph
ESC w	Copies region into kill ring without removing region in current buffer

To mark a region for manipulation in emacs, merely position the cursor to the beginning of the region and press:

```
C-@
```

or

```
C-SPACE
```

whichever is most convenient. Once this mark has been set, you can move the cursor (also known as the point) to the end of the region, using any emacs command. Once you have reached the end of the region, there is nothing special that you have to do. emacs assumes the region begins with the mark and ends with current cursor position. You can verify the position of the mark by issuing the command:

```
C-x C-x
```

This will place the cursor (point) at the current mark position and will place the mark position at the previous cursor location. This is useful to remind you exactly what you are deleting. Note, however, that it is not necessary to do this. At this point you can delete the region with the command:

```
C-w
```

Remember that you can undo your deletion with the C-y command.

To copy a block of information, you can mark the region as described before and, once the region is marked, issue the command:

```
ESC w
```

This copies the current marked region to the kill ring. Next, move the cursor to the desired location and issue the C-y (yank) command. This will insert the contents of the last deletion stored in the kill ring in the current location.

You can manipulate the contents of the kill ring with commands like:

```
ESC y
```

which deletes the most recent text in the kill ring. The kill ring is essentially a last in first out (LIFO) queue which acts much as a traditional stack operates. With commands like those listed above you can manipulate the next block of information in the kill ring to be operated on. For example, you can use the ESC y command to remove the last

item from the kill ring until you are at the item you wish to yank with C-y.

The default size of the kill ring is the last 30 deletions. This can be modified with the command:

```
ESC x set-variable RETURN kill-ring-max RETURN value RETURN
```

where value is the number of deletion items to save in the kill ring.

9.14 Using Multiple Buffers

You can also use multiple buffers in emacs to move blocks of information around. Each buffer has a major mode associated with it which determines its state. You can have any number of buffers each containing any amount of information. You can move around within buffers and move between buffers at will. You can merge and cut buffers from one to another and save the resultant buffer to a disk file. There is almost no limit to what you can do with buffers in emacs.

Buffers are a good way to save some but not all information from one file to another. Let's take an example of how we could do this. We have a file named test1 which contains the following information:

```
This is line 1
This is line 2
This is line 3
This is line 4
This is line 5
```

To invoke emacs on this file, type:

```
$ emacs test1
```

To cut lines 3 and 4 out of this file, issue the commands, position the cursor at the beginning of line 3, and press:

```
C-SPACE
```

This marks the first line of the region to be copied. Move the cursor to the beginning of line 4 and press ESC w. This copies lines 3 and 4 to the kill ring. Now you need to open a new buffer. Use the command:

```
C-x b buffername
```

where buffername is the name of the new buffer you want to create. Note that it is probably a good idea to have the buffer name match the filename. Once you type this, a new buffer is created and a blank

screen is presented. To yank the last contents of the kill ring into the current buffer, type:

```
C-y
```

which yanks the last kill ring contents to the current buffer. From here you can save and manipulate this buffer as you would any other.

You can list the buffers in your session with the command:

```
C-x C-b
```

This produces a screen which contains information on the current list of buffers and associated information while leaving a portion of the screen as your current buffer. To move to the buffer list, use the command:

```
C-x o
```

Once you are in the buffer list, there are a variety of commands which you can issue to act upon buffers in the list. Some of the more basic are:

`C-n`	Moves down a line
`C-p`	Moves up a line
`SPACE`	Moves to the next buffer in the list
`dx`	Marks the buffer for deletion
`sx`	Saves the buffer
`1`	Displays the buffer
`2`	Displays the buffer in a portion of the window

9.15 Halting the Execution of emacs Commands

You can halt the execution of any emacs command with the command:

```
C-g
```

This will place you back in input mode and stop the execution of the current command.

9.16 Saving a File

To save a file, issue the command:

```
C-x C-s
```

This will save the contents of the current buffer to a file of the same name. If you have any problems with terminal hangs, they are prob-

ably related to the C-s sequence, which may stop flow control to your terminal. If this occurs, type a C-q to unlock the screen and allow data to begin to flow again.

If you want to save the buffer contents as something other than its buffer name, issue the command:

```
C-x C-w
```

and type the filename under which to save the buffer.

9.17 Exiting emacs

To exit emacs type:

```
C-x C-c
```

emacs will ask you if you want to save any unsaved changes. Answer y or n as appropriate.

9.18 Suspending emacs

You can temporarily suspend emacs with the command:

```
C-z
```

This is a function of the job control of the system and will not work on all systems. If you successfully return to the shell prompt, you can reenter the emacs editor at exactly the place you left by typing:

```
$ fg
```

You can also use the standard job control commands to control more than one background process.

9.19 Autosave Files

emacs, by default, creates an autosave file every 300 keystrokes. The name of the autosave file is simply the original filename preceded and followed by # marks. For example, if you issue the command:

```
$ emacs kevin
```

after the first 300 keystrokes a file named #kevin# would be saved. This will be the place, if a crash occurs, you can recover from. You can

force a flush of the current buffer back to the last autosave with the command:

```
ESC x revert-buffer
```

You will then be asked whether you want to revert to the most recent autosave buffer (y or no) and then if you want to revert buffer from file filename (yes or no). This means that you can either go back to the last complete group of 300 keystrokes or to the beginning of your edit session. Note that emacs expects exact matching answers, so type y, n, yes or no, respectively.

emacs also creates a backup file when you begin an edit session on an existing file. The file is named the filename followed by a ~. For example, if you entered the command:

```
$ emacs kevin
```

and kevin existed, a file named kevin~ would be created which contained the contents of kevin before any changes were made. This allows you to keep a backup copy in case you want to disregard any committed changes.

9.20 Multiple Windows in emacs

emacs provides for multiple window support on most platforms. While many workstations have moved to bitmap support and use windowing systems, you may want to split a current window into multiple screens. From within each screen you can perform any task you could from any full-function emacs screen. For example, you can issue e-mail and shell commands and can edit other files.

The basic command to split the screen is:

```
C-x 2
```

This will split you current window into two segments, as shown in Fig. 9.5. While you can use a mouse and windowing system capabilities to create multiple windows, it is often more efficient to simply issue the above command and perform the task at hand than finding your mouse, creating a new window, entering the editor, etc.

To move from one window to another, use the command:

```
C-x o
```

To delete a window, type:

```
C-x 0
```

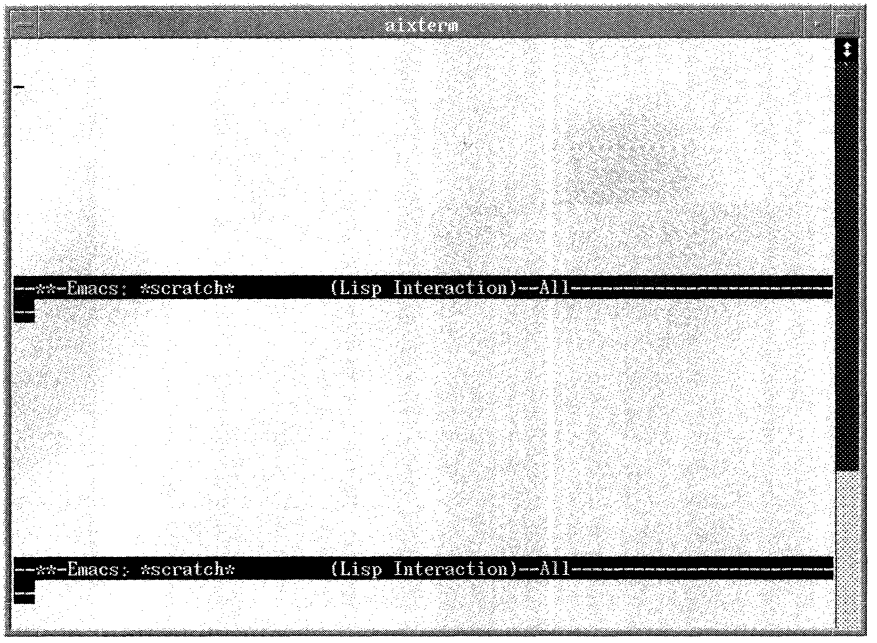

Figure 9.5 emacs split screen.

Note you have all the consequences and behaviors of each of the split windows being a full-function editing session. Keep this in mind when you are killing buffers and moving back and forth between them.

You can alter the size of windows within a screen with the commands:

`C-x ^`	Makes the current window taller
`C-x {`	Makes the current window wider

These commands alter the current window size by one line.

9.21 Searching for and Replacing Text

Every editor has a basic search and replace capability; emacs, as you are probably coming to expect, has a variety of search and replace capabilities. They range from simple searches to complete regular expression searches. Keep in mind that the flexibility of emacs allows you to write your own search and replace algorithms, and this may be something you want to do when you have become more familiar with emacs.

The basic search categories are:

`C-s ESC string`	Simple search forward
`C-r ESC string`	Simple search backward
`C-s string`	Incremental search
`C-s ESC C-w string`	Exact string search
`ESC x re-search-forward`	Regular expression search (see below)
`ESC x replace-string RETURN oldstring RETURN newstring`	Search and replace (see below)
`ESC % RETURN oldstring RETURN newstring RETURN`	Query-replace

There is commonality between all search command syntax. All commands begin with a control sequence such as C-s (for forward searches) or C-r (for backward searches) and end with a carriage return. All commands can also be repeated to find the next occurrence by pressing either the C-s or C-r command sequence without entering a string. Finally, to end any string search or replace operations, type ESC. This tells emacs you are finished with any search and replace operations and clears the buffer from which to search. It is important to type ESC when you are finished searching to ensure that you are not continuing the actions of the last executed search command.

Remember also that emacs works from the current cursor position, and therefore if you want to execute commands from the beginning of the file, you should position yourself at the beginning of the file and before executing any necessary commands.

Finally, emacs is case independent. When you issue commands related to case, emacs ignores it for the purpose of search and matches it for the purpose of replace. In other words, emacs attempts to match case character-by-character when replacing text so as to minimize the side effects of the word change. For example if you have a file which contains:

```
I have a file which has several occurrences of oldstring. Oldstring
really consists of a bunch of gibberish which really has no meaning
or context. I hope the OLDSTRING in all caps will not confuse
anything I do in the emacs editor.
```

and you issue the command:

```
ESC x replace-string RETURN oldstring RETURN newstring
```

the result would be a file which contained:

```
I have a file which has several occurrences of newstring. Newstring
really consists of a bunch of gibberish which really has no meaning
or context. I hope the NEWSTRING in all caps will not confuse
anything I do in the emacs editor.
```

As you can see, the emacs editor attempts to maintain case on a character-by-character basis. This is useful, particularly when programming since languages such as C are case sensitive.

9.21.1 The simple search

The simple search consists of entering a string and telling emacs to find the next occurrence. For example, to find the next occurrence of the string emacs, use:

```
C-s ESC emacs
```

This will find the next forward occurrence in the file from the current insertion point. emacs may be a string inside a longer string and the match will still occur. If the string gnuemacs occurred next, the cursor would be positioned at the e in the string gnuemacs. The exact string match will look for a unique occurrence of the string emacs if that is what you are after.

To repeat the forward find, type the command:

```
C-s
```

To do a backward find, issue the command:

```
C-r ESC emacs
```

and emacs will search backward from the current position for any occurrence of the string emacs. To repeat the backward find, issue the command:

```
C-r
```

9.21.2 The exact string search

If you are looking for a unique word and not a string, you need to use the exact string (or word) match capability of emacs. To search for the word emacs, issue the command:

```
C-s ESC C-w emacs
```

To find the next occurrence, issue the command:

```
C-s
```

You can also issue backward searches with the C-r version of this command. For example, to find the previous occurrence of the emacs word, issue the command:

```
C-r ESC C-w emacs
```

and to repeat the find, issue the command:

```
C-r
```

9.21.3 The incremental search

The incremental search allows you to enter a string, and emacs will begin matching the string with the first letter. This means that you can continue to narrow the search for your match by entering characters in the string you wish to match.

To match the first string that begins with a, type:

```
C-s a
```

The first occurrence of the character a will become the current position. If you are looking for the occurrence abc, you would simply type a b and a c and watch as emacs continues to narrow the results of your search. This command is most useful to programmers since they may not remember the exact string but know how it begins in the program.

Remember to type ESC when you are finished with your search and replace activity since this will ensure that you are not telling emacs something you don't intend. Note that you can also end the search with another control sequence. Since emacs is CTRL command driven, this is normally how the search is ended; keep in mind that if you don't issue an ESC or CTRL command sequence after you have finished searching, emacs assumes you are still in search mode and will act as such.

9.21.4 Search and replace

Many times you are interested in replacing one or all occurrences of a string in a given file. To replace all occurrences of a word in your current file, type:

```
ESC x replace-string RETURN oldstring RETURN newstring RETURN
```

Remember that the ESC x sequence tells emacs to execute a typed command. In this case the command is replace-string and the oldstring will be replaced with the newstring in all occurrences of the file. Remember also that emacs works from the current position and only those occurrences following the cursor position will be changed.

9.21.5 Query search

If you want to replace only certain occurrences of a string, you must use the query search capability of emacs. To enter a query search, type the command:

```
ESC % RETURN oldstring RETURN newstring RETURN
```

Once emacs finds the first occurrence of oldstring, the following question appears at the bottom of the screen:

```
Query replacing oldstring with newstring.
```

At this point, emacs waits for you to tell it whether to replace the string or perform any of a variety of other options on the string. Some of the available options are:

DEL	Doesn't replace; moves to next occurrence of oldstring
n	Same as DEL
SPACE	Replaces and moves to next occurrence of oldstring
y	Same as SPACE
.	Replaces current occurrence and quits
,	Replaces current occurrence but doesn't go to next occurrence
^	Moves back to previous occurrence
!	Replaces all following occurrence and doesn't ask for assistance
ESC	Quits query replace

By far the most common answer is SPACE (space bar) or y. Note that UNIX and emacs are case sensitive and therefore Y is not the same as y. Keep this in mind if you have any troubles in emacs.

9.21.6 Regular expression searches

Regular expressions are typically used when you want to perform a general substitution from a predetermined pattern. Through the use of wildcard substitutions, you can generate strings which can be matched according to your specifications. emacs can use regular expressions to locate strings and words in any of the basic types of search and replace operations. For example, to find a word that is at the end of a line, issue the command:

```
C-s ESC C-w last$
```

This will search for the next occurrence of the word last that is the last word on a line. The most basic characters that can be used in regular expressions are:

*	Matches any number of characters
.	Matches a single character
$	Matches the end of line
^	Matches the beginning of line

The regular expression commands react similarly to those described earlier. The commands for this should be invoked from the command-level interface. The basic syntax is:

`ESC x re-search-forward`	Simple search regular expression
`ESC x re-search-backward`	Simple search regular expression
`ESC x isearch-forward-regexp`	Incremental serach regular expression
`ESC x isarch-backward-regexp`	Incremental search regular expression
`ESC x query-replace-regexp`	Query replace regular expression
`ESC x replace-regexp`	Replace regular expression

With these commands, you can execute all the commands described above but have the added flexibility of using regular expressions to match multiple strings with substitution capabilities.

9.22 Text Formatting and Its Relation to emacs

emacs has several modes which support a variety of text formatting utilities. Tools such as troff, nroff, and TeX are supported within emacs. You can use emacs to assist you with writing these types of files, and it provides basic syntax checking and generation for each of the text formatting languages described above.

There are high levels of support for troff and nroff and exceptional levels of support for TeX and LaTeX. emacs provides syntax checkers and generators for these markup languages and, in addition, processes them into files which can be processed at a later time if necessary.

Each of the major text formatters is represented by a major mode in emacs. This gives you the ability to simply enter a major mode and have emacs assist you with formatting and syntax. For example, the most common type of standard text formatting system on UNIX is nroff. This is the format which man pages and some Internet documentation come in. Therefore, some of the basic capabilities of the nroff major mode in emacs will be discussed. It should be noted that nroff mode is very similar to troff, and most of the commands will work in both modes.

To enter emacs major mode, type:

```
ESC x nroff-mode
```

You can now use certain defined behaviors in the nroff major mode to assist you with moving around in the file, formatting the file, and with

syntax. For example, you can use emacs to automatically insert the ending part of a macro. To place emacs in the mode known as electric-nroff-mode, type the command:

```
ESC x electric-nroff-mode
```

An example of this is shown in Fig. 9.6.

When you type .PP, you then press C-j. This creates a duplicate .PP with a blank line in between and positions the cursor on the beginning of the blank line. From here you can type your paragraph text. Another example is to type .LG press C-j; the corresponding .NL will be placed below with a blank line in between. This function is very useful for building macros in nroff or troff format in the nroff major mode of emacs.

You can also use emacs to assist with moving around in the file. Because nroff uses a special file format which does not include blank lines, you may have trouble moving around in the file easily. nroff major mode provides commands which move you both forward and backward with respect to text files. They bypass the interceding macro commands. They are:

`ESC n`	Moves forward to next text line
`ESC p`	Moves backward to next previous text line

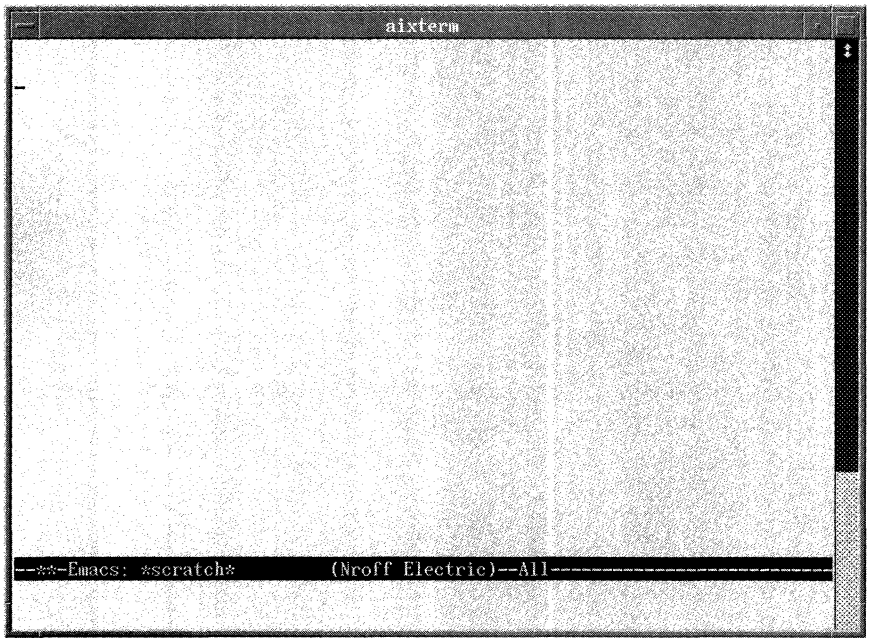

Figure 9.6 electric-nroff-mode.

With these commands, you can save yourself quite a bit of time by not having to move, one line at a time, over all macro commands to get to the next line of text.

There are other commands in the nroff mode; see the on-line help for more information.

9.22.1 TeX and LaTeX support

Similar support exists for TeX and LaTeX. The two major modes are:

`tex-mode`	emacs determines whether file is TeX or LaTeX and places you in the correct mode.
`latex-mode`	emacs is placed in LaTeX mode.
`plain-tex-mode`	emacs is placed in TeX mode.

emacs will verify curly braces and quotes as well as provide comment capabilities.

emacs also provides additional support for TeX in the form of processing. You can process the contents of your TeX or LaTeX input file with the command:

```
C-c C-b - process current buffer
```

All output messages from this command are placed in the TeX shell buffer. This should come up automatically; if it doesn't, press:

```
C-c C-l
```

This will process your file and produce a dvi file (device independent file). You can then process the dvi file and send it to the default printer with the command:

```
C-c C-p
```

This will queue the processed dvi file to the default printer. To check the queue status, use the command:

```
C-c C-q
```

Finally, to stop processing of a TeX or LaTeX file, type the command:

```
C-c C-k
```

9.23 Shell Commands

You can escape directly from emacs into a shell window with the command:

```
ESC x shell
```

This forks a new shell and places you at the command prompt. From here you can issue any shell commands as you normally would. To exit the shell, type:

```
$ exit
```

This places you back in emacs at the place you left.

You may only want to execute a single command and reenter the emacs editor at the conclusion of this command. To do this, type:

```
ESC ! command
```

emacs opens a window with the results of this command. To kill the window, type:

```
C-x 1
```

Because the output of the command is in a buffer, you can perform any emacs command on the results from within emacs.

9.24 emacs Customization

emacs is one of the most customizable editors available today. By learning a small amount of LISP, you can create your own command languange and essentially build your own editor interface. Some companies have gone so far as to structure editor commands to emulate an old editor from which they are migrating in an effort to minimize the migration pain for the developers. This has been very successful, at least in terms of their ability to create an editor which fits their needs.

There are many variables in emacs which control almost every aspect of its behavior. The primary way to change the state of variables is with the command:

```
ESC x set-variable RETURN variable RETURN value RETURN
```

The set-variable command allows you to enter a variable name and a subsequent value for that variable. This command works in almost every instance and allows you to customize emacs to behavior that you expect.

9.25 X Windows Support

There are several newer versions of emacs which support X Windows. Namely, 18.59 has support for X Windows. Support is determined when you build the emacs product. You can choose to provide X Windows support when the emacs executable is built.

If you choose to provide X Windows support, you gain the ability to create multiple windows and utilize the full power of emacs. You will know if X Windows support is included by simply invoking the emacs editor from an xterm window (or some derivative). If a new window appears, you are running with X Windows support; if it doesn't, you probably aren't. Talk to your system administrator for more details.

9.26 Spell Checking

You can perform basic spell checking on the current word in emacs with the command:

```
ESC $
```

emacs processes the current word and prompts for an action. The action list is the same as that presented for the query search capability in emacs. You would typically type one of:

RETURN	Word is correct; store in the dictionary.
SPACE	Begins a query replace response to correct word.
C-g	Quits without changes.
n	Goes to next occurrence without changes.

To check the spelling of an entire file, use the command:

```
ESC x spell-buffer
```

This is one of the few commands that will begin its execution at the beginning of the buffer without respect to the current cursor position. It will scan through the entire file, stopping at each occurrence of a word it does not know and asking for a response as described above.

9.27 Printing from within emacs

You can issue certain commands to print from within emacs. Some of the most basic are:

print-buffer	Prints contents of current buffer to default printer
lpr-buffer	Prints contents of current buffer without any preprocessing by lpr
print-region	Prints contents of current region

You invoke the above commands as any other emacs command:

```
ESC x command RETURN
```

You can configure emacs to issue the print commands with certain characteristics with the command:

```
ESC x lpr-switches -Pibm_laser RETURN
```

This sets the output queue to ibm_laser. This can be any combination of switches you normally place on the lpr command.

9.28 Other Things You Can Do in emacs

There are many things you can do within emacs such as invoke mail and execute commands. While this chapter has discussed how to invoke commands, it will not discuss mail since this interface, while functional, is not as sophisticated as most mail interfaces today.

One interesting thing you can do is read manual pages and nroff documents in general from within the emacs editor. To read a man page, issue the command:

```
ESC x manual-entry RETURN command RETURN
```

This will bring up the manual page for command. To exit, simply exit the buffer as you normally would.

9.29 emacs and Programming Languages

Just as emacs supports several text formatters, it also supports and understands several programming languages. Languages with their own major modes include C, Fortran, LISP, and others. This book will only discuss C and Fortran since they are the main languages in use with emacs today. You can also create your own language support modes with the autoload command. See interactive help for more information on this macro.

9.29.1 C language support

emacs has support for automatic recognition of mode based on file extensions. For example, emacs will place you in C mode if you read in a file with an extension of .c, .h, or .y. emacs has an understanding of how the C syntax looks and can assist you with things like semicolons, curly braces, and quotation marks. It can also help you format your code so that you can read it more clearly months or years after you wrote it. To enter this mode manually, type:

```
ESC x c-mode
```

You can preserve indentation of your code by simply pressing LINEFEED (C-j) instead of RETURN at the end of a text line. emacs will then indent to the first character of the previous line. This is very useful when writing in a structured language such as C and saves keystrokes and time.

You can also format an entire region with the command:

```
ESC C-\
```

Simply mark the beginning of a region as described earlier in this chapter, place the cursor at the end of the desired region, and press the above sequence. This will indent all intervening lines appropriately. Note the use of the word *appropriately.* emacs has some knowledge of the form and syntax of C and will make indentations based on blocks and structures (curly braces, conditional loops, etc.) which significantly increase the readability of the code. You can alter the indentation defaults for C source code with the standard ESC x command macro. Some of the more basic variables available are:

`c-argdecl-indent`	Indentation for type declarations of functions (default 5)
`c-auto-newline`	Inserts a newline before and after {} and after ; and : (default nil)
`c-brace-offset`	Indentation for line that begins with { (default 0)
`c-continued-statement-offset`	Extra indentation on continuation lines (default 2)

There are other variables. See the on-line help for more information. Many programmers alter these with C since it can save a tremendous amount of time and resources as you key in the code itself.

You can insert comment lines either at the end of a current line of text or at the beginning of a blank line with the command:

```
ESC ;
```

This inserts the correct comment line syntax (for C it is a /* */) structure and places the cursor in the blank part of the comment string. Use this to place a large number of comments in your code since this will increase maintainability.

There are other commands which move you within your source code file. Some of the more basic are:

`ESC C-a`	Moves to beginning of function body
`ESC C-e`	Moves to end of function body

These allow you to jump between functions, bypassing the rest of the body with one key sequence.

9.29.2 etags (external tags)

There is another function in emacs related to moving between functions quickly and easily. With the etags command, you can generate an etags file which contains function references generated from a listing of files. This is useful when your program is contained in more than one source code file. Instead of being forced to move between files and buffers, you can generate a tags file which contains file and function mappings for emacs. This allows you to act on a function name which is not in the current buffer and automatically reference that function and move it into the current screen.

You must invoke etags as a separate command with a syntax of:

```
etags -f outfile infile ...
```

where -f outfile saves the output tag file in the file outfile. The default is a file named TAGS.
infile ... is one or more .c, .h, or, .y files which contain information related to your C program.

You can use wildcards with any file on the etags command line just as you can with any other command. One possible invocation of etags is the following:

```
$ etags *.[chy]
```

This will build the TAGS file which contains all function references in all files in the current directory which end in .c, .h, and .y. You could create a TAGS file with the command:

```
$ etags *.[chy] /usr/local/kevin/*.[chy}
```

to create a TAGS file from all possible C source files in both the current directory and the /usr/local/kevin directory.

Once you have created a TAGS file, you can act on it from within emacs with the following commands:

`ESC x visit-tags-table RETURN filename RETURN`	Reads in the tag file.
`ESC . [tag]`	Where tag is the function you want to find. If you don't enter a tag, emacs will use the word the cursor is currently on.
`ESC ,`	Finds the next occurrence of tag.
`ESC x list-tags`	Generates a list of all tags in current file.
`SC x tags-appropos`	Generates a list of all matched strings.

`ESC x tags-query-replace`	Allows a search and replace, much as described in an earlier section.
`ESC x tags-search`	Allows you to enter a regular expression on which to search.

9.29.3 Fortran language support

Just as with C, there is support for the Fortran language in emacs. Many of the functions are similar and, as you will see, are invoked in much the same way.

When you invoke emacs with a filename which contains an extension .f, you will be place in Fortran mode by default. By selecting a major mode of Fortran, emacs provides support for syntax and structural checking, much the same as is supported for C. Some of the most basic commands are:

`ESC ^`	Joins current line to the previous one
`ESC C-j`	Splits current line at cursor position
`ESC C-a`	Moves to beginning of current subprogram
`ESC C-e`	Moves to end of current subprogram
`C-c C-n`	Moves forward to next statement
`C-c C-p`	Moves backward to previous statement
`C-c C-w`	Creates window 72 columns wide to aid in entering source

There are also variables similar to those supported by C. Some of the most basic are:

`fortran-do-indent`	Additional indentation used in do statements (default 3)
`fortran-if-indent`	Additional indentation used in if statement (default 3)
`fortran-line-number-indent`	Indentation of line numbers (default 1)
`fortran-minimum-statement-indent`	Positions beginning of statement (default 6)

With respect to line numbers in Fortran, you can simply type the line number at the beginning of the line, and emacs will move your cursor to column 7 or the column indicated by the variable fortran-minimum-statement-indent.

9.29.4 LISP language support

LISP is a language which is supported by a major mode of emacs. Because emacs is written in LISP, there is strong support for syntax and development of LISP in emacs. Using LISP and its relation to emacs is beyond the scope of this book; however, suffice it to say that emacs' support of LISP may well be its best supported languages. See related books and on-line help for more information.

9.29.5 Compiling programs

One of the main benefits for programmers in emacs is the ability to compile code from within emacs itself. The basic command to invoke the compilation system is:

```
ESC x compile
```

This will prompt you with a series of questions relating to the compilation process. The default command is:

```
make -k
```

where k continues with compilation after errors (this is not make default).

You can enter your own compilation command which becomes the default for the session by using the compile command as described above.

emacs gives you the ability to move from one error to the next with the command:

```
C-x `
```

When you issue the above command, emacs places your cursor where the error occurred. You can then fix the problem and either move to the next error by again issuing a C-x ` or reissue the compilation command with ESC x compile.

Each time you issue the C-x ` command, you are placed on the next error message in the Compilation buffer. To begin at the first error again, use the command:

```
C-u C-x `
```

You can then cycle through the error messages again, using the standard C-x ` command.

9.30 Multiuser File-Level Locking Support

One of the problems with editing in a UNIX environment is that there is no support for multiple users editing the same file at the same time. Because of the nature of UNIX, this kind of logic was not built into the file management system. This can cause problems with editors like vi since two or more people can edit the same file at the same time and the one who saves the file last will overwrite any changes made by the other users editing the file.

emacs supports file-level locking such that when a user begins to edit

a file, if another person is already editing this file, a warning message will be issued notifying you that there are others editing the same file. This can occur either when you read in the file or attempt to save the file. Either way you are made aware of others editing the same file that you are working on. This kind of functionality is critical to business and is one of the most powerful and important features of emacs in a business environment.

9.31 Conclusion

emacs is quite simply the most powerful and extensible editor available today. While the commands will initially seem awkward, you will quickly get productive and soon grow to love the emacs editor. Most professional UNIX people, particularly those who work on multiple platforms and architectures, use emacs for all editing tasks.

While there are other tools which may provide a better partial solution, it is difficult to find a tool which provides more power and flexibility than emacs. With its all encompassing environment, including syntax checking, structure assistance, mail capabilities, and compilation capabilities, emacs provides a unified environment for working in UNIX. Once you enter emacs, you may not ever have to leave to accomplish your goals. This is the real power of emacs.

There are many emacs features which have not been discussed in this chapter. See the on-line help or other books for more information. See the recommended book list in App. D for more information on possible books.

Chapter

10

Nonnative Output Formatting and Display Tools

There are many native tools in UNIX which provide basic text formatting and display capabilities. Most of these tools are based on output for a teletype since this is how UNIX began. Because of this, most native UNIX text formatting tools are somewhat primitive by today's standards.

This chapter outlines several tools which provide significantly enhanced function to a UNIX platform and associated text processing. These packages also work on most UNIX and non-UNIX platforms and provide portable input and output files. This becomes very useful when you want to generate documents for more than one computer architecture.

Other tools provide the capability to preview documents both in a standard tty-type environment and in a windowing environment such as X11. This saves time and paper when building drafts of documents which are large. Keep in mind that there are several standard text processing tools which come with UNIX, and the tools documented in this chapter merely enhance the overall text processing capabilities of most UNIX machines.

A tool called Tcl is discussed. Tcl is a very powerful scripting development language which adds significant power to the AIX development environment, particularly in the area of GUI development. Check the Internet for more information.

Other tools such as xloadimage are discussed, which will allow you to directly manipulate various types of images such as GIF, TIFF, and JPEG directly from within your windowing environment. Have fun.

10.1 Ghostscript

10.1.1 Introduction

Ghostscript is a language that very closely resembles Postscript. It contains interpreters and drivers for many machines and operating systems such as:

IBM PC and compatibles running DOS with EGA, VGA, or SVGA

Many UNIX systems running X11R3, X11R4, and X11R5

Apple Macintosh

Sun workstations running Sunview

VAX/VMS running X11R3, X11R4 or X11R5

Ghostscript provides for multiprotocol display of Postscript-like files across heterogeneous platfoms and displays. It provides output drivers for a variety of cards and display devices as well as Postscript-compatibles devices such as printers and plotters. You can preview documents with the Ghostscript previewer before printing to make any necessary corrections.

The primary interface to Ghostscript is the Ghostscript interpreter, which allows you to display, print out, or save interpreted Ghostscript files. The Ghostscript interpreter provides several options to control both input interpretation and output. Many Postscript files will work with this interpreter. The advantage of Ghostscript is not only its cost (*free*) but its multiplatform capability.

Ghostscript was written by GNU but is actually maintained and distributed by Aladdin Enterprises. Aladdin can be reached at:

Aladdin Enterprises
P.O. Box 60264
Palo Alto, CA 94306
(415)322-0103
...{uunet,decwrl}!aladdin!ghost
ghost@aladdin.com

Aladdin suggests that you subscribe to the Usenet newsgroup gnu.ghostscript.bug. Because of Ghostscript's widespread distribution, Aladdin makes no claims to support the product fully but will respond to e-mail if they have time. Aladdin is looking for help with modifications to Ghostscript. See the README file for more information. Aladdin also sells commercial licenses of Ghostscript and associated products. Call or write them for more information.

See the readme.doc and NEWS files for information on up to the minute fixes and enhancements to the product.

Ghostscript is a GNU product and as such is subject to its GNU Gen-

eral Public License as included both in the product distribution and in App. C of this book.

10.1.2 Installation

The version of Ghostscript that was used to generate this chapter was 2.6.1. Check for the current version using information provided in Chap. 6.

As is standard with most Internet software packages for UNIX platforms, the build procedures for Ghostscript are contained in the makefile. There is a file called make.doc which describes what needs to be changed within the makefile for Ghostscript to build properly. There is also a file called use.doc which describes how to use the Ghostscript interpreter. There are other doc files which describe different aspects of the Ghostscript distribution. They are:

`README.RS6k`	Descriptions of files in Ghostscript distribution
`drivers.doc`	Describes the interface between drivers and Ghostscript
`history.doc`	Describes the history of product releases
`humor.doc`	Humorous comments on Ghostscript
`fonts.doc`	Information about fonts in distribution
`language.doc`	Description of Ghostscript language
`lib.doc`	Information about Ghostscript libraries
`make.doc`	Describes installation and configuration issues
`man.doc`	Manual page for Ghostscript
`psfiles.doc`	Information about .ps files included
`readme.doc`	Contains information regarding features
`use.doc`	Information about how to use Ghostscript

First make a copy of the compressed tar file in a directory other than the one in which you are building Ghostscript. This will allow you to start over if things look like they are going in the wrong direction.

Once you have gotten the compressed tar file from the Internet, unwind the tar file into the appropriate directory and begin to build the executables. Ghostscript unwinds the files into one directory known as ghostscript2.6.1. While this is not a good practice, it does allow for easiest distribution since all files are in one place. You can modify the structure after you build the product. Note that if you are unfamiliar with or unsure how to extract, uncompress, and build an appropriate directory structure for Ghostscript, see Sec. 6.9 for more information.

Examining and modifying the makefiles. There are many makefiles in the Ghostscript distribution. A brief listing of the ones of interest to UNIX users are:

`bc.mak`	Initial makefile for MS-DOS/Borland C++ platform
`bcwin.mak`	Initial makefile for Windows 3.*x*/Borland C++ platform
`gc.mak`	Generic makefile used for all platforms
`devs.mak`	Makefile listing all device drivers
`ansihead.mak`	Initial makefile for ANSI C compilation
`cc-head.mak`	Initial makefile for Kernighan & Ritchie C compilation
`devs.mak`	Makefile for device drivers including printers and displays
`fonts.mak`	Initial makefile for Ghostscript fonts
`gcc-head.mak`	Initial makefile for GNU C compilation
`gs.mak`	Generic Ghostscript makefile
`msc.mak`	MS-DOS msc 7.0 makefile
`vs-aix32.mak`	Makefile for UNIX/ANSI C/XII configurations
`tc.mak`	Initial makefile for Turbo C platform
`unix-ansi.mak`	More of the makefile for ANSI C compilation
`unix-cc.mak`	More of the makefile for K&RC compilation
`unix-gcc.mak`	More of the makefile for GNU C compilation
`unixhead.mak`	Generic part of makefile for all UNIX C compilations
`unixtail.mak`	Generic part of makefile for all UNIX C compilations

The makefiles are structured using variables. This allows you to simply change the variable at the beginning of the makefile and the rest of makefile can remain unchanged. The areas that may need to be changed are:

Default search paths for fonts and initialization files

Debugging options

Device drivers to be included

Optional features to be included

The defaults for UNIX machines are:

Current directory where product source code exists

No debug code included

Devices are platform specific

Features are platform specific

Each makefile contains an initial comments section followed by a section entitled Options. The Options section is where you may want to make edits. Do not make edits outside the Options sections without understanding exactly what you are doing.

The first step to building Ghostscript is determining which series of makefiles you are going to use. There are three basic sets of makefiles revolving around different C standards and compilers. K&R C is the old standard, which was the original C syntax. Most C compilers still

support this syntax. If you have a compiler which supports only the K & R standard, use the command:

```
$ln -s unix-cc.mak makefile
```

This creates a symbolic link called makefile. The make utility looks for a file makefile by default. If you are using the GNU C compiler (known as gcc), you should issue the command:

```
$ln -s unix-gcc.mak makefile
```

Finally, if you can, you should use an ANSI standard C compiler. Most compilers, including gcc, support the ANSI standard. If you are using an ANSI standard compiler but not gcc, issue the command:

```
$ln -s unix-ansi.mak makefile
```

You must also be concerned with the location of the X11 libraries which are linked to create the Ghostscript executable. The default location for these libraries is /usr/local/include. If this is not the correct location, you must change the XINCLUDE macro on invocation of the make command or modify the proper makefile for your compiler.

There are issues relating to the X11 server and screen refreshing methods and tiling. This behavior should be okay. If you have problems, consult the make.doc for more information on some possible fixes.

For specific machine notes, consult the make.doc. An example of building Ghostscript 2.6.1 on an RS/6000 running AIX 3.2 is as follows:

```
$ ln -s rs-aix32.mak makefile
$ make XCFLAGS=-D_POSIX_SOURCE -DSYSV\
> XINCLUDE=-I/usr/lpp/X11/include XLIBDIRS=-L/usr/lpp/X11/lib
```

You may get a link error on the final link. Try simply typing make with no options; the link seems to work correctly that way (your guess is as good as mine). Note that this is different from the notes in the build.doc file in that the path to the include and library files for X11 are different. This is one of the most common mistakes that must be corrected between machines. The X11 files are in different places on different UNIX implementation, and you should use the XINCLUDE and XLIBDIRS macros to set the correct directory. According to the Ghostscript documentation, people often still have trouble on the RS/6000.

You can also use gcc to build Ghostscript with a command like:

```
$ ln -s unix-gcc.mak makefile
$ make CFLAGS= XINCLUDE=-I/usr/lpp/X11/include
XLIBDIRS=-L/usr/lpp/X11/lib \
GENOPT=SVR4
```

This will build Ghostscript with gcc. Note that because of the way linking is done, you should make sure you have used gcc to create X11 libraries if you are going to use gcc to create Ghostscript. If you have not done this, you should use the vendor compiler as documented in the first example. Note that the linking conventions may be different between the gcc and vendor C compiler, and you may experience problems with symbol resolution. If you do, try the vendor compiler.

This is an example of how to build a software product and the kinds of things you typically run into. Keep in mind that compiler differences and X11 library and include files are often 95 percent of the battle when porting to a platform. Once you get by these problems, you are off and running. There are several machine-specific comments in the gs.1 manual page. Use nroff -man to preview this and ensure that you have any machine-specific issues understood before you begin to build Ghostscript. See the file README.RS6k for additional information.

10.1.3 Usage

To invoke the Ghostscript interpreter, type the command:

```
$ gs [filename...]
```

Other available options are listed below. The interpreter looks for the existence of several initialization files such as:

`gs_*.ps`	Initialization files which configure initial screens for Ghostscript, including gs_init.ps, gs_fonts.ps, gs_statd.ps, etc.
`Fontmap`	Contains fontsmaps for all fonts used by Ghostscript

Once the interpreter has processed the files on the input command, it waits for input from standard input, which is the keyboard. The interpreter will interactively interpret the commands you type in. To quit the interpreter type exit.

As is the standard for most UNIX applications, to get help, invoke the interpreter with a -h or -?. For example:

```
$ gs -h
$ gs -?
```

When you invoke Ghostscript, it opens a window on your display by default using the X11 protocol. Of course, you must be running an X11 windows-compliant window manager and GUI. This means that you can distribute the display of the Ghostscript files across the network to any machine that supports the X11 protocol. In addition to the display window, the original invoking window becomes the Ghostscript control window. You are asked to press a return to continue. If you press the

return, the display window is cleared and you are placed at the Ghostscript (GS>) prompt. From this prompt you can enter interactive commands that conform to the Ghostscript language for immediate interpretation on the display device.

For example, type:

```
$ gs tiger.ps
```

Another window will open on your screen with a color picture of a tiger's head (see Fig. 10.1). Once the picture is drawn, Ghostscript pauses and waits for you to do something. Typing a return will clear the display window and place you at the interactive prompt. For more

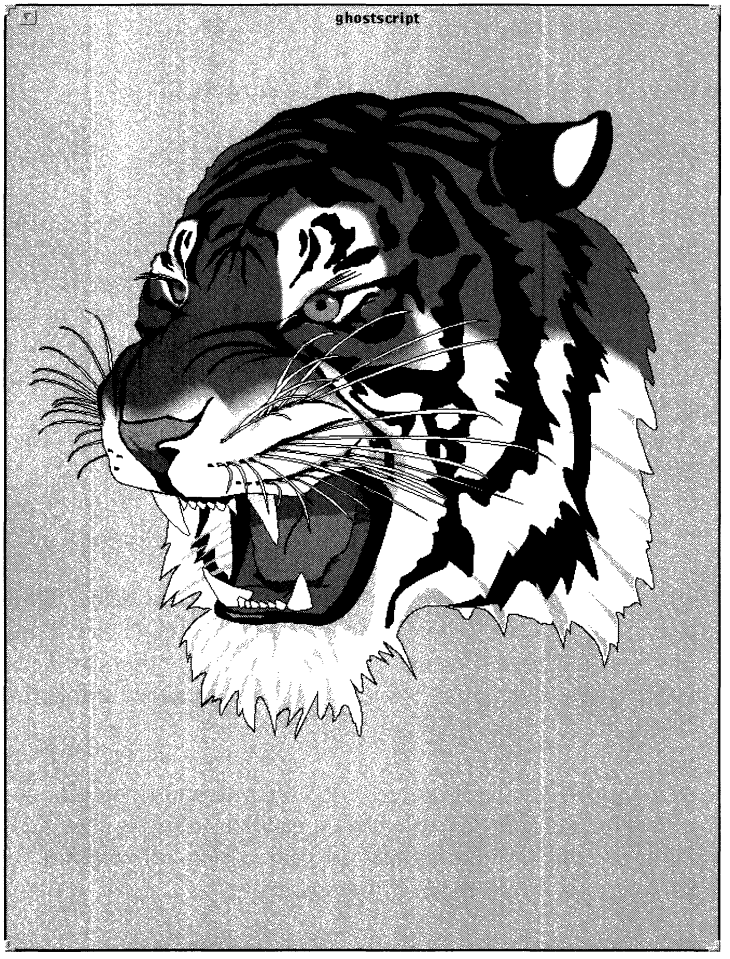

Figure 10.1 The Ghostscript display.

information on some of the commands you can type, see the section below on the Ghostscript language.

You input the filename, and Ghostscript searches the current directory, then any specified by the -I switch, any specified by the GS_LIB environmental variable, and, finally, directories specified by the GS_LIB_DEFAULT macro in the Ghostscript makefile.

There are several parameters which can be set in your .Xdefaults file. They are:

`borderWidth`	Border width in pixels (default 1)
`borderColor`	Border color (default black)
`geometry`	Window size and placement (format WxH+X+Y, where W is width in pixels, H is height in pixels, X is numbers of pixels from left hand side of screen and Y is number of pixels from top of screen)
`xResolution`	Number of x pixels per inch
`yResolution`	Number of y pixels per inch

You can place these in your .Xdefaults file in the format:

```
Ghostscript*geometry: 1280x1024+0+0
```

This will make the display window cover your entire display if you have a screen capable of displaying 1280 × 1024 resolution. It probably makes sense to use something a little bit smaller but feel free to experiment. Note also that it is not necessary to use the xrdb command to load these characteristics into the X11 server as documented. This makes these characteristics available to all users of the X11 server. This may or may not be what you want to do.

There are several other switches available to Ghostscript; however, the only one that is useful to the general user is:

```
-dNOPAUSE
```

which disables the prompt and pauses at the end of each page. This allows other applications to use Ghostscript to drive their output. This means that you no longer have to press return to clear the display window and end the display of a page. See the use.doc for more information.

Devices. You can choose multiple devices for output from the Ghostscript interpreter. The default for the UNIX environment is the X11 display. There is a switch on the command line which allows you to select the output device on invocation of Ghostscript:

```
-sDEVICE=devicename
```

where devicename is:

sonyfb (monochrome Sony display)

sunview (Sunview window system)

X11 (X11R3 or greater)

Any of a variety of printers (see the devs.mak for more information) and a variety of graphical ouput formats such as GIF, PCX, and bitmap formats

An alternative is to set the environmental variable GS_DEVICE to the device you would like to use. Ghostscript will look for this variable on invocation and use this as the default output device. As a result of the build, device files are built with an extension of .dev. If you examine all *.dev files in your Ghostscript directory, you will see which devices are supported on your system. This is determined in part by which makefile you choose for your make procedure. Look for the existence of the *.dev files in your Ghostscript directory. To see what devices names are availble for your configuration, type:

```
GS> devicenames ==
```

Additional drivers can be inserted and built for specific devices you own.

Manual pages. There are two manual pages distributed with the product: man.doc and man1.doc. The man.doc file contains a very basic man page which can be used to describe syntax, etc. The man1.doc file is a much more complete man page and should be used for the manual page for users of the system. The man page is in nroff an macro format. This means that to view the man page, you should use the command:

```
$ nroff -man man1.doc | more
```

Note that the nroff command interprets the man1.doc file and uses the an macro for assistance. The pipe to more lets you see it exactly as it would appear through the man system. If you have questions concerning the structure of the man system or how best to use the man pages from Internet software, see Sec. 6.6.

A brief discussion of the Ghostscript language. Ghostscript supports Postscript Level 1 and many of the Display Postscript extensions. It also supports many of the Level 2 operators. It is an interpreter for a language specification that is described in the *Postscript Language Reference Manual* (Addison-Wesley, 1985).

Some of the very basic commands allow for file manipulation such as:

`GS> (filename) run`	Where filename is a Postscript file
`GS> devicenames ==`	Reports what devices are supported
`GS> (devicename) selectdevice`	Where devicename is from list above

The Postscript language itself is a very sophisticated language which contains commands to perform almost any task you could expect from a graphics language. See the standard Postscript documentation above for more information about commands available at the GS> prompt. Also see language.doc for more information.

10.1.4 Conclusion

The Ghostscript interpreter is a very powerful graphics display and interpretation tool that runs on many platforms and is available free of charge from a variety of sources. Full Postscript support and interpretive capabilities make Ghostscript a very powerful tool for viewing documents and graphics before committing them to paper.

10.2 Ghostview

10.2.1 Introduction

Ghostview is a full-function user interface for Ghostscript. While Ghostscript provides the ability to preview Postscript files by interpreting the Postscript input files, Ghostview provides an interactive interface which allows you to manipulate the Ghostscript and provides a much more interactive and user-friendly interface. In summary, Ghostview creates and manages the X11 interface for Ghostscript.

Ghostview is a GNU product and as such is subject to its GNU General Public License as included both in the product distribution and in App. C of this book.

10.2.2 Installation

There is an Imakefile distributed with Ghostview, and you can use this to generate the appropriate makefile for your particular platform. See Sec. 7.17 for more details on imake. You may have to modify the IRULESSRC macro in the Imakefile or the makefile if you use imake. This will point imake to the proper template directories.

If you don't have imake installed, you can edit the makefile manually. The areas you may have to modify are primarily the directories which point to the X11 subdirectories. For example, on an RS/6000 if

the xmkmf command does not create the correct makefile, you want to define:

```
LIBSRC=/usr/lpp/X11/lib
LIBDIR=/usr/lpp/X11/lib
```

You can use the xmkmf for the Sun, but this doesn't typically ship with xmkmf for AIX. Simply find the DEFINES definition and add:

```
DEFINES = -DSYSV ...
```

This ensures that the system is built as a System V system, which most closely emulates the RS/6000.

There is also a file named Makefile.rs6k which is delivered. You can make this your makefile and Ghostview should build correctly.

As with many of the delivered makefiles, the defaults are structured according to the standard SVR4 X11 structure, which is /usr/X11/lib and in actuality, exists on very few machines. Keep this in mind when you examine the makefile.

Or, if you recreate the appropriate makefile with imake, you can simply issue the command:

```
$ make
```

This will build the software system. If you have errors, examine them and make the appropriate modifications to the makefile and reissue the make command. Finally, to install the resulting files, issue the command:

```
$ make install
```

10.2.3 Usage

Ghostview provides much the same usage interface as does Ghostscript. The basic syntax is:

```
ghostview [--[no]install] [-[no]private] [-[no]center] [-[no]title]
[-[no]date] [-[no]locator] [-resolution dpi] [-dpi dpi] [-xdpi dpi]
[-ydpi dpi] [-[no]quiet] [-preload file] [-magstep n] [-portrait]
[-landscape] [-upsidedown] [-seascape] [-letter] [-tabloid]
[-ledger] [-legal] [-statement] [-executive] [-a3] [-a4] [-a5] [-b4]
[-b5] [-folio] [-quarto] [-10x14] [-force] [-forceorientation]
[-forcemedia] [-[no]swap] [-[no]openwindows] [-[no]ncdwm]
[-page lable] [file]
```

where -install installs standard color map.
-private installs nonstandard color map.
-center centers the page in the viewport (default).

-title displays the %%title comment (default).
-date displays the %%data comment (default).
-locator displays locator (default).
-resolution dpi sets display resolution at dpi dots per inch.
-dpi dpi is the same as -resolution.
-xdpi dpi, -ydpi dpi sets x and y resolution at dpi dots per inch.
-quiet doesn't produce informational messages (default).
-preload file preloads file which may include fonts, etc.
-magstep n defines document magnification. The formula used to display the document on the screen is (1.2*magstep) and values range from –5 to 5.
-portrait displays in portrait mode.
-landscape displays in landscape mode.
-upsidedown displays upside down.
-seascape displays in seascape mode (this is counterclockwise 90 degree rotation).
-letter displays page in 8.5 × 11 in.
-tabloid displays page in 11 × 17 in.
-ledger displays page in 17 × 11 in.
-legal displays page in 8.5 × 14 in.
-statement displays page in 5.5 × 8.5 in.
-executive displays page in 7.5 × 10 in.
-a3, -a4, -a5 displays page in 842 × 1190, 595 × 842, and 420 × 595 Postscript points, respectively.
-b4, -b5 displays page in 729 × 1032 and 516 × 729 Postscripts points, respectively.
-folio displays page in 8.5 × 13 in.
-quarto displays page in 610 × 780 Postscript points.
-10 × 14 displays page in 10 × 14 in.
-force tells Ghostview that orientation or media is being forced and may not be the default.
-forceorientation tells Ghostview that orientation is being forced.
-forcemedia tells Ghostview that media is being forced.
-swap swaps the meaning of landscape and seascape and use the %%Orientation comment in the input source file.
-no openwindows turns off bitmaps usage.
-no ncdwm ignores bug in window resizing when running the ncdwm.
-page label labels the displayed page with label.
file is input file to be processed.

Most of the options discussed above are the defaults. Note that most options contain opposites which can be enabled by either including or excluding the no prefix to the option. Most of the options above will

never be used, but they will be useful if you want to change the default behavior of your Ghostview or X11 environment.

Ghostview fully supports the Adobe Document Structuring Conventions which control things like page size and orientation. The basic window is shown in Fig. 10.2. There are five buttons in the window: File, Page, Magstep, Orientation, and Media. Each is a pull-down button which provide specific function related to a major operational area of Ghostview. The File button provides all file-level interaction with Ghostview. The Page button gives you the ability to move around within the pages of a Postscript document. The Magstep button allows you to magnify the document to allow for easier viewing or high resolu-

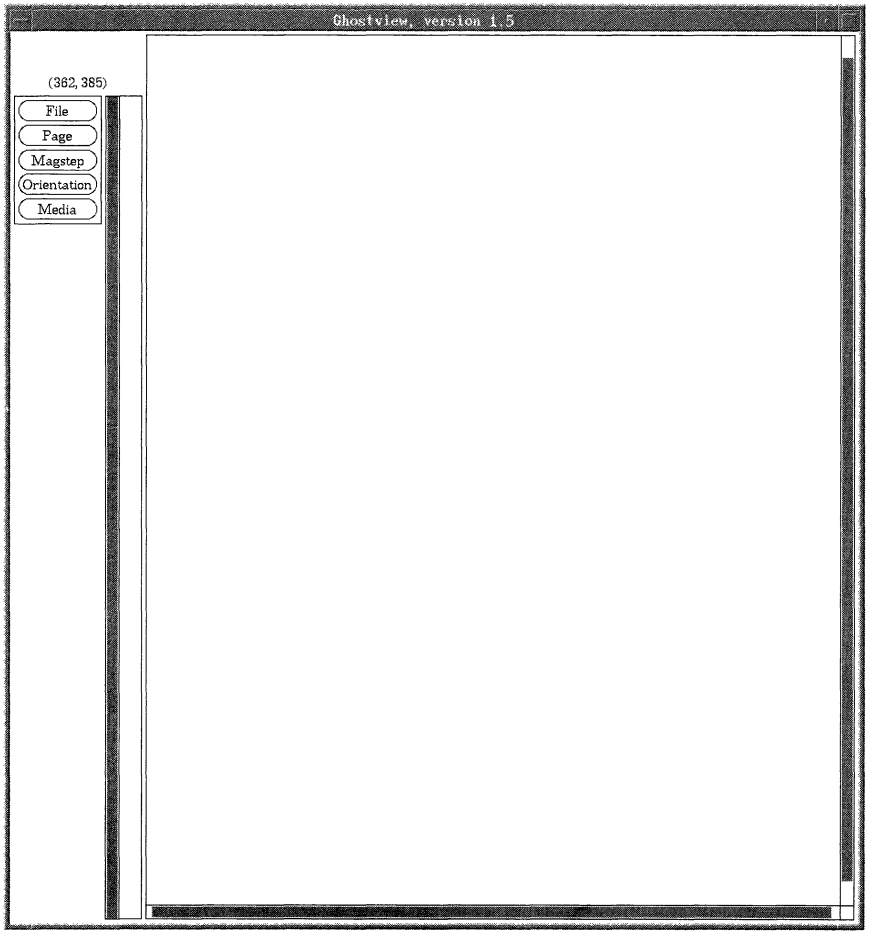

Figure 10.2 The Ghostview screen.

tion of the document. The Orientation button allows you to change the display of the document from landscape to portrait, to flip the document, and to have other orientations of the page. Finally, the Media button provides a list of possible mediums for display including Letter, Tabloid, Ledger, Legal, and others. There are also keystrokes called Keyboard accelerators which provide faster access to commands than the mouse. Once you have used Ghostview a few times, you will want to use these to speed your interaction with the product. See the ghostview.ps file for more information on these options and their explicit meanings and operations. There is a section on FAQs in the README file. See this if you experience any problems or have questions about a certain behavior.

Because Ghostview is an interactive interface to Ghostscript, there are several ways you can control the integration of the two tools. By setting the environmental variable GHOSTVIEW to the X11 window id of the window you want Ghostscript to use, you cause Ghostscript to draw on an existing window instead of creating its own separate window. This is useful when you want to concentrate all graphics in one window and use the attributes such as size and orientation of the current window to display the graphic you are looking at.

10.2.4 Conclusion

Ghostview is a very user-friendly inteface to Ghostscript. While Ghostscript has a variety of cryptic commands with which you display and manipulate commands, Ghostview provides a single command interface which displays documents using the capabilities of Ghostscript. Based on experience, Ghostview is the primary interface you will want to use for displaying and manipulating information on your display.

10.3 groff

10.3.1 Introduction

groff is the GNU equivalent of troff and nroff. It provides support for much of the command set contained in the nroff and troff languages as well as for all devices supported by these tools. groff also provides support for additional devices including X11 previewers and Postscript output devices. In addition, groff provides portability across multiple platforms, including many varieties of UNIX as well as VMS and other operating systems. This implementation of groff also contains GNU versions of troff, pic, eqen, tbl, refer, and a variety of macros, including an and mm. See the README file for more details on the groff distribution itself.

10.3.2 Usage

The basic syntax of the groff command is:

```
groff [-tpeszaivhblCENRVXZ] [-wname] [-Wname] [-mname] [-Fdir]
[-Tdev] [-ffam] [-Mdir] [-dcs] [-rcs] [nnum] [-olist] [-Parg]
[file...]
```

where -t preprocesses with gtbl.
-p preprocesses with gpic.
-e preprocesses with geqn.
-s preprocesses with gsoelim.
-z supresses output from groff.
-a—see gtroff below.
-i—see gtroff below.
-v displays version number.
-h displays help.
-b—see gtroff below.
-l sends output to the printer.
-C—see gtroff below.
-E—see gtroff below.
-N doesn't allow newline with eqn commands.
-R preprocesses with grefer.
-V prints pipeline on standard output.
-X previews with gxditview.
-Z doesn't postprocess output.
-wname, -Wname, -mname, -Fdir—see gtroff below.
-Tdev processes output for device dev.
-ffam, -Mdir, -dcs, -rcs, -nnum, -olist—see gtroff below.
-Parg passes arg to the postprocessor.
file ... is file or files to process.

Many of the options for groff are the same as those for nroff and troff. Available devices consist of Postscript devices, dvi format for TeX processors, X75 for 75 dpi X11 previewer, and X100 for X100 dpi X11 previewer. To get more information on all available devices, see the directories named devname where name is the name of the device class.

The commands are the same as those used by nroff and troff, and, as such, you should see the man pages for nroff and troff.

groff also contains an nroff implementation. The basic syntax is:

```
gnroff [-hi] [-mname] [--nnum] [-olist] [-rcn] [-Tname] [file...]
```

where all options are describe in the gtroff doc section below.

gtroff is the fundamental package contained and used by most other tools in the groff distribution. The basic syntax of gtroff is:

```
gtroff [-abivzCER] [-wname] [-Wname] [-dcs] [-ffam] [-mname] [-nnum]
[-olist] [-rcn] [-Tname] [-Fdir] [-Mdir] [file...]
```

where -a generates ASCII output.
-b displays a trace with each error message.
-i reads standard input after all input files are processed.
-v displays version.
-z supresses formatted output.
-C enables troff compatibility mode.
-E suppresses error messages.
-R doesn't load troffrc.
-wname enables warning name.
-Wname disables warning name.
-dcs defines c as a string s.
-ffam uses fam as the default font family.
-mname reads in the file tmac.name.
-nnum numbers the first page num.
-olist outputs comma-separated pages in list.
-rcn sets number register c to n.
-Tname outputs device format.
-Fdir searches dir for directories containing devname where name is the device name.
-Mdir searches dir for macro files.
file... is one or more files to process.

gtroff works very similarly to troff and has support for more and newer devices such as X11 previewers and high-resolution Postscript printers.

gtroff does have some features in addition to those in troff. Support for fractional point sizes, numeric expressions including boolean logic operators such as < and >, and new escape sequences such as \A'anything', which becomes 0 or 1 depending on the properness of the value of anyting. There are other escape sequences which support new gtroff funtionality. gtroff also supports a variety of different commands which manipulate fonts, number registers, provide conditional expressions, and do many other things. See the file troff/troff.man for more details. Remember, to view this file before you have built gtroff, use a command like:

```
$ nroff -man troff/troff.man | more
```

This will display the output just as the man command would.

Another powerful command included in the groff distribution is the grog. The basic syntax is:

```
grog [-options] [file ...]
```

where -options are gtroff options which will be inserted to the results of the grog command.
file ... is one or more files to process.

The grog command stands for groff guess. If you run this command on a groff input file, grog will guess which macros need to be used to print the file or files and will insert them into the groff commmand option automatically. This is very useful when you are not sure what macros are used in a file and simply want grog to tell you and execute the appropriate command.

There are also subdirectories which contain commands such as refer, which builds a bibliography for groff; eqn, which builds equations; tbl, which builds tables; grotty, which generates standard tty output; and grops, which generates Postscript output. The best way to see what commands are available is to examine the subdirectory structure within the groff directory with a command like:

```
$ ls -l | grep ^d | more
```

This will display and page a listing of the directories within the current working directory.

The other useful way to see the commands in the current groff distribution is to use a command like:

```
$ find . -name "*.man" -print
```

This will generate a listing of all man pages in the current directory structure.

There are a variety of macros and environmental variables which determine the execution charactersistics of groff. By settings variables such as GROFF_TMAC_PATH, GROFF_PATH_DIR, and FONTDIR, you can change the way gtroff works. See the associated man pages for each tool to understand which environmental variables you can change to change gtroff output and behavior.

10.3.3 Installation

The first requirement for the installation of groff is a C++ compiler. This will probably mean the g++ compiler, which is documented in Sec. 7.2. Besides g++, you will need the libg++ libraries to provide all class

header information so that the compilations and links will proceed normally. g++ 2.5 and later contain all the necessary header files to build grott and therefore it is not necessary to install libg++. See Sec. 7.2 for more details on installing the g++ compiler if you need it.

Once you have installed the C++ files and compiler, you can run the standard configure install, which will build the appropriate makefiles, which can then be executed. The basic format is:

```
$ ./configure --prefix=/usr/local
$ make clean
$ make
$ make install
```

If you have problems with the make, examine the beginnings of the makefile for the macro definitions which may be incorrect. There are a variety of makefiles spread throughout various subdirectories. You may want to investigate gmake as an alternative to make to support the VPATH variable and various path distributions. The configure will attempt to see that you have the correct files and headers for your C++ compiler before creating the makefile. If you have trouble with the configure, make sure you have installed and made available all compiler and header/include files for g++ (gcc) before invoking configure.

There is a simple command to test the build of groff:

```
$ test-groff -man -Tascii groff/groff.n | more
```

This will display reasonable looking output on the display. If it doesn't, something hasn't built correctly. See the makefiles for more details. If you have other problems, see the file named PROBLEMS for potential known problems and solutions.

Be sure to set all appropriate environmental variables such as FONTDIR when you build the product. See the PROBLEMS sections for more details if you have problems. Once the product is built, you shouldn't have any problems; however, if you do, it is probably a misdefined environmental variable. See the INSTALL and PROBLEMS files for more details.

Remember also that there are makefiles in each subdirectory which describe the dependencies for that particular tool.

Finally, there is tool included called gxditview. This is an X11 previewer based on xditview. To build this product, you must cd to the xditview subdirectory and issue the make command. There is an INSTALL file which describes this process in more detail in the xditview subdirectory. As with most X11-related tools, there is an associated Imakefile which may need to be tweeked and remade with the xmkmf command. Once you have done this, rebuild the product with the make

command. This will generate the appropriate gxditview executable. If you continue to have problems, check the FONTDIR, XINITDIR, and XDMDIR macro definitions to ensure that these point to the correct directories. Often these files do not exist in the standard /usr/.../X11 directory. This is key to generating a correct build and is classic "gotcha" when building X11-related tools.

10.3.4 Conclusion

groff is a very powerful environment which consists of nroff and troff functionality as well as the associated tools eqn, restor, tbl, and others. See the files in doc and man for more details on these commands. groff also comes with a groff X11 previewer called gxditview, which can be built separately to provide bitmapped graphical capabilities.

groff is clearly one of the most powerful environments available on the Internet and should be investigated if you are doing any work with text processing and output formatting.

10.4 pbmplus

10.4.1 Introduction

plmplus is a tool kit which consists of a variety of filters and conversion programs to take one file format and generate another. There are also some tools to manipulate various portable file formats.

The distribution is broken down into functional units consisting of:

1. PBM—bitmap manipulation
2. PPM—full-color image manipulation
3. PNM—content-independent manipulations

All units are backward compatible, which means that PNM supports both PPM and PBM, and PPM supports PBM. Which units you install is determined when you install the product. See Sec. 10.5.3 for more details.

pbmplus is a free product and as such is subject to its own copyright and license as included both in the product distribution and in App. C of this book.

10.4.2 Usage

The basic usage of the pbmplus package is driven by individual commands contained within the package. The basic mode of operation for pbmplus is to take a given input format and generate a standardized

generic output format. From this format you can generate any other given format with a different tool. pbmplus has four generic (or intermediate) formats:

1. pbm (bitmap)
2. pgm (grayscale)
3. ppm (pixmap)
4. pnm (any format map)

where the bitmap format represents a pixel with a bit, while the grayscale has additional information to preserve shading and, finally, the pixmap format preserves color information about each pixel. The pnm format supports all three previous formats as well as a variety of others. Each of these generic formats has a set of associated tools which manipulate these formats and generate the desired output format.

There are manual pages for each of the commands which convert from one format to another. All man pages end with a .1 and exist in the pnm, pbm, pgm, and ppm subdirectories. Because of the large number of man pages and the small number of options to these commands, they are not documented here. The basic format of all commands is:

```
$ command file
```

where command is the conversion or filter program and file is the file to be converted.

There are really two basic kinds of utilities included in pbmplus. The first kind contains conversion and filter programs which take one kind of information and convert it to another. The second kind of utility programs manipulates pnm format files. These programs are typically used once you have converted a file into pnm format to do things like crop the picture size, enhance a bitmap, and rotate and scale a bitmap. Keep in mind that these operate on the pnm format files which have already been converted to pnm format. Both types of utilities are described in separate tables below.

The following table outlines the conversion and filter commands and their basic functionality.

Format	To	From
Abekas YUV bytes	tuvtoppm	ppmtoyuv
Andrew Toolkit raster object	atktopbm	pbmtoatk
ASCII files		pbmtoascii
Atari degas .pi1	pi1toppm	ppmtopi1
Atari degas .pi3	pi3topbm	pbmtopi3

Format	To	From
Atari compressed spectrum file	spctoppm,	
Atari uncompressed spectrum file	sputoppm,	
Bennet Yee face file	ybmtopbm	pbmtoybm
bitgraph graphics		pbmtobg
CMU window mgr bitmap	cmuwmtopbm	pbmtocmuwm
DEC sixel		ppmtosixel
Doodle brush	brushtopbm,	
Epson printer		pbmtoepson
FITS	fitstopgm	pgmtofits
GEM .img file	gemtopbm	pbmtogem
GIF	giftoppm	ppmtogif
Gemini 10X printer graphics		pbmto10x
Gould scanner file	gouldtoppm,	
GraphOn compressed graphics		pbmtogo
Group 3 fax	g3topbm	pbmtog3
HIPS	hipstopgm,	
HP Laserjet		pbmtolj
HP Paintjet	pjtoppm	ppmtopj
IFF ILBM	ilbmtoppm	ppmtoilbm
Img-whatnot	imgtoppm,	
LISP Machine bitmap	lispmtopgm	pgmtolispm
MGR bitmap	mgrtopbm	pbmtomgr
MacPaint	macptopbm	pbmtomacp
PICT	picttoppm	ppmtopict
Motif UIL icon file		ppmtouil
MTV or PRT ray tracer output	mtvtoppm,	
NCSA ICR format		ppmtoicr
PCX	pcxtoppm	ppmtopcx
Portable bitmap		pgmtopbm
Portable graymap		ppmtopgm
Postscript image data	psidtopgm,	
Postscript data		pnmtops
Printronix printer graphics		pbmtoptx
QRT ray tracer output	qrttoppm,	
Raw RGB	rawtoppm,	
Raw grayscale	rawtopgm,	
Sun icon	icontopbm	pbmtoicon
Sun rasterfile	rasttopnm	pnmtorast
Text	pbmtext,	
Three portable graymaps	rbg3toppm	ppmtorgb3
TIFF	tifftopnm	pnmtotiff
TrueVision targa file	tgatoppm	ppmtotga
UNIX plot file		pbmtoplot
Unknown		anytopnm
Usenix FaceSaver	fstopgm	pgmtofs
X10 bitmap	xbmtopbm	pbmtox10bm
X11 window dump	xwdtopnm,	
X11 puzzle file		ppmtopuzz
X11 bitmap	xbmtopbm	pbmtoxbm
X11 pixmap	xpmtoppm	ppmtoxpm
X11 window dump	xwdtopnm	pnmtoxwd
Xim file	ximtoppm,	
Zinc bitmap		pbmtozinc

There are also utilities included with the pbmplus release which operate on these generic bitmap-type files, as listed in the table below. With these utilities you can enhance, resize, reshape, and reformat the generic bitmap files. It should be noted that many of the techniques used in these utilities are taken from *Beyond Photography* by Holzmann and *Digital Image Processing* by Gonzalez and Wintz. See the individual man pages for more information on the algorithm and its origin.

Utility	Function
pbmlife	Applies Conway's Rules of Life to portable bitmap
pbmmake	Creates a blank bitmap
pbmmask	Creates a mask bitmap
pbmreduce	Reduces portable bitmap n times
pbmupc	Creates a universal product code bitmap
pgmbentley	Utilizes the Bentley effect to smear the bitmap
pgmedge	Edge detects the graymap
pgmenhance	Edge enhances a portable graymap
pgmhist	Creates a histogram of the graymap
pgmnorm	Normalizes the contrast of the graymap
pgmoil	Creates an oil painting (smearing technique)
pgmramp	Creates a grayscale ramp (useful with other bitmaps)
pnmarith	Performs arithmetic on two anymaps
pnmcat	Concatenates anymaps
pnmconvol	Generates M×N convolution on anymaps
pnmcrop	Crops an anymap
pnmcut	Cuts a rectangle out of an anymap
pnmdepth	Changes pixel depth on an anymap
pnmenlarge	Enlarges an anymap
pnmfile	Describes an anymap
pnmflip	Flips an anymap in any direction
pnmgamma	Performs gamma correction on an anymap
pnmindex	Builds an index of several anymaps
pnminvert	Inverts an anymap
pnmmargin	Adds a border to an anymap
pnmnoraw	Converts anymap to plain format
pnmpaste	Pastes a rectangle into an anymap
pnmrotate	Rotates an anymap by a given angle
pnmscale	Scales an anymap
pnmshear	Shears an anymap by a given angle
pnmsmooth	Smooths out an anymap
pnmtile	Duplicates an anymap in a specified size
ppmdither	Dithers a color image
ppmhist	Builds a histogram of a pixmap
ppmmake	Creates a pixmap of a specified size and color
ppmpat	Creates a "pretty" pixmap
ppmquant	Quantizes colors down to some chosen number
ppmquantall	Operates on multiple files so they can share a colormap
ppmrelief	Runs a laplacian relief filter on a pixmap
sxpm	Shows and/or converts XPM2 files to XMP3 files

Assume you have a G3 fax file named kevin.g3 and you want to create a Postscript file of it. You would issue the commands:

```
$ tifftopnm kevin.g3 > kevin.pnm
$ pnmtops kevin.pnm > kevin.ps
```

Note that most, if not all, of the commands generate their output to standard output. You can redirect standard output as you normally would to create the new file.

With respect to the utility programs, they are typically more complex in their syntax and require a brief glance at the man page associated with the command. For example, to use the pnmenlarge command, you might use something like:

```
$ pnmenlarge 2 kevin.pnm > newkevin.pnm
```

This will effectively double the size of the kevin.pnm bitmap file. You can then create your Postscript file as before with the command:

```
$ pnmtops newkevin.pnm > kevin.ps
```

This generates the Postscript file with an image twice as large.

Other utility commands use different syntax. See the man pages for more information. As mentioned in the earlier sections, you can view a man page with the command:

```
$ nroff -man pnmtops.1 | more
```

This is exactly what the man command does, so have at it.

As you can see from the tables above, there are many commands from which to choose and , therefore, many file formats are supported.

10.4.3 Installation

The installation of pbmplus consists of a variety of makefiles, each existing in a directory that contains source files. The main directories are pbm, pgm, ppm, and pnm. These contains all the utilities and filters related to the corresponding type of input file format.

Within each makefile, there is a section marked CONFIGURE. These are the sections you must modify to ensure that the build proceeds correctly. For example, ld and cc command options are described, each in their own CONFIGURE section. You can also define which directories will contain the output files and which tools (such as gcc) are used to configure and build the pbmplus system.

The first step in building pbmplus is building the included tiff libraries. Use the following series of commands to build libtiff:

```
$ cd libtiff
$ make clean
$ make -f makefile.aix
```

This will create all necessary files in the libtiff directory. Note that several makefiles for different platforms are included in this directory. Choose the AIX makefile for the RS/6000. Note that your compiler may or may not support newer C functions such as prototypes, and you may have to change variables such as -DPROTOTYPE=1 to -DPROTOTYPE=0 to disable prototyping capabilities. The other macros you may be interested in are USE_VARARGS and BSDTYPES. See the associated makefile for the correct definition of these macros for your particular platform.

There are a variety of macros which you can add to the definitions within the makefile. These are documented both in the README file in the libtiff subdirectory and in the makefile in the libtiff subdirectory itself. Note that these affect whether tags for JPEG and other algorithms are compiled into the system. Choose the default as the first step to simplify the build process. There is nothing to stop you from going back later and rebuilding the pbmplus distribution with different build flags.

Once you have built the libtiff subdirectory, you are ready to build the rest of the software in the pbmplus distribution. Modify each relevant CONFIGURE section in the makefile in the main pbmplus directory. Once this is finished, type the command:

```
$ make
```

This will generate the appropriate executable files which match those described in the two tables above. Note that if you are using gmake, you may have to modify each makefile in each of the pxm subdirectories (where x can by b, g, p, or n). See each makefile for more details. It is recommended that you use non-GNU make to build this tools since it introduces less complexity to the situation than gmake in this case.

If you have any trouble, you may need to run xmakefile to build a new makefile. Once you have done this, proceed as described above.

10.4.4 Conclusion

pbmplus is a very powerful tool kit which provides conversion and utility programs for many of the bitmap file formats available today. By using the tools in pbmplus, there is virtually no file format that you cannot support in full graphical form. By using the ability to convert from one format to many others, you can suddently share information from a variety of mediums transparently. This is clearly a very powerful UNIX tool kit.

10.5 gnuplot

10.5.1 Introduction

gnuplot is a command-driven plotting package which contains a variety of functions and commands to generate plots on a large number of devices. You can automatically generate labels, define constants and functions, and manipulate plots from an interactive interface. gnuplot is one of the most powerful plotting packages available for UNIX today.

gnuplot is a GNU product and as such is subject to its GNU General Public License as included both in the product distribution and in App. C of this book. It does have a slightly different copyright notice than the usual GNU Copyright, and therefore it has its own notice in App. C.

10.5.2 Usage

The basic syntax of the gnuplot command is:

```
gnuplot [X11 options] [-mono] [-gray] [-clear] [file ...]
```

where X11 options consist of many of the standard options you would expect to use with an X11 application such as border size, image size, color maps, and aspect ratios.
-mono uses monochrome rendering.
-gray uses grayscale rendering.
-clear clears the device between plots.
file ... is one or more files to plot.

gnuplot supports two basic types of plotting terminals, X11 and x11, where the X11 device supports a different points plotting style than x11.

You can see the man page on X for more details on standard X11 options. Most of these can be applied to the gnuplot utility.

gnuplot has its own interactive language which is case sensitive and emulates the UNIX command syntax. You can use the \ to move beyond one line and the ; to separate multiple commands on a single line.

The basic types of output devices supported by gnuplot are:

AED	AED 512 and AED 767
AIFM	Adobe Illustrator Format
AMIGASCREEN	Amiga custom screen
APOLLO	Apollo graphics
ATT6300	AT&T 6300 graphics
BITGRAPH	BEN bitgraph
CGI	SCO CGI
CORONA	Corona graphics

DXY800A	Roland DXY800A plotter
DUMB	Dumb terminal
DXF	AutoCad dxf file format
EEPIC	EEPIC LaTeX driver
EGALIB	EGA/VGA PC graphics
EMTEX	LETeX with emTeX specials
EPS60	Epson 60dpi printers
EPSONP	Epson LX-800, Star NL-10, NX-1000, etc.
FIG	Fig graphics lnguage
GPR	Apollo graphics
HERCULES	Hercules graphics board
HP2648	HP2648, HP2647
HP26	HP2623A
HP75	HP7580
HPGL	HP7475 and other GL printers
HPLJII	Laserjet II
HPLJII	Laserjet III
IMAGEN	Imagen laser printers
IRIS4D	Iris computer
KERMIT	Kermit Textronix 4010 emulator
LATEX	LaTeX picture environment
LN03P	DEC LN03P printer
NEC	NEC CP6 printer
PBM	PBMPLUS pbm, pgm, ppm, and pnm
POSTSCRIPT	Postscript
PRESCRIBE	Kyocera printer
QMS	QMS/QUIC printer
REGIS	DEC ReGis graphics
SELANAR	Selanar
STARC	Star color printer
SUN	Sun workstation
T410X	Textronix 4106, 4107, 4109, and 420x terminals
TANDY60	Tandy DMP-130 series printers
TEK	Tektronix 4010
UNIXPC	AT&T Unix PC
UNIXPLOT	Unixplot
V384	Vectrix 384 color printers
VTTEK	VT-like Tektronix 4010 emulator
X11	X11R4 window system

If you examine the README file, you will see that there are a variety of commands, probably the most important being the help command. From within gnuplot, you can issue the help command to see a listing of all commands available. Rather than duplicate all information given in the interactive help section, below is a table of commands and their basic function.

`autoscale`	Automatically scales axis to incorporate all data.
`bugs`	Displays a list of current bugs.
`cd`	Changes the current working directory.
`clear`	Clears the current screen.
`comments`	Describes comment support.
`environment`	Describes a number of environmental variables which can be set.
`exit`	Leaves gnuplot.
`expressions`	Describes expression support.
`help`	Describes this help facility.
`line-editing`	Describes basic line-editing capabilities.
`load`	Inputs a file for processing.
`pause`	Causes a pause of a specified time or until the return is pressed.
`plot`	Displays the plot.
`print`	Prints expressions to the screen.
`pwd`	Prints working directory.
`quit`	Exits gnuplot.
`replot`	Redraws the current plot.
`save`	Saves user-defined functions or variables to a file.
`set`	Sets a variety of options which are described in this help.
`shell`	Forks and execs to a shell.
`show`	Shows the values of set commands.
`splot`	Plots 3-D information.
`startup`	Defines the start-up file (.gnuplot).
`substitution`	You can use command line substitution within gnuplot with ".
`userdefined`	You can define your own functions and variables within gnuplot.

See the individual help sections for more information on each command. The basic things you need to understand to use gnuplot effectively are plot and expressions.

Expressions consist of functions which are supported by C, BASIC, Fortran, and other third-generation languages. This means that you can use C- or Fortran- or BASIC-like syntax to define a function to plot on the display or print to a printer. An example of a very simple expression to display to the screen is the command:

```
gnuplot> plot cos(x)
```

Note, however, that if you do this without either setting the TERM environmental variable or using the set term command, you will get an error message. The best way to do this is something like this.

```
gnuplot> set term dumb
gnuplot> plot cos(x)
```

This will display a cosine curve on a dumb device such as a vt100 or other ASCII device. The output will look something like that shown in Fig. 10.3.

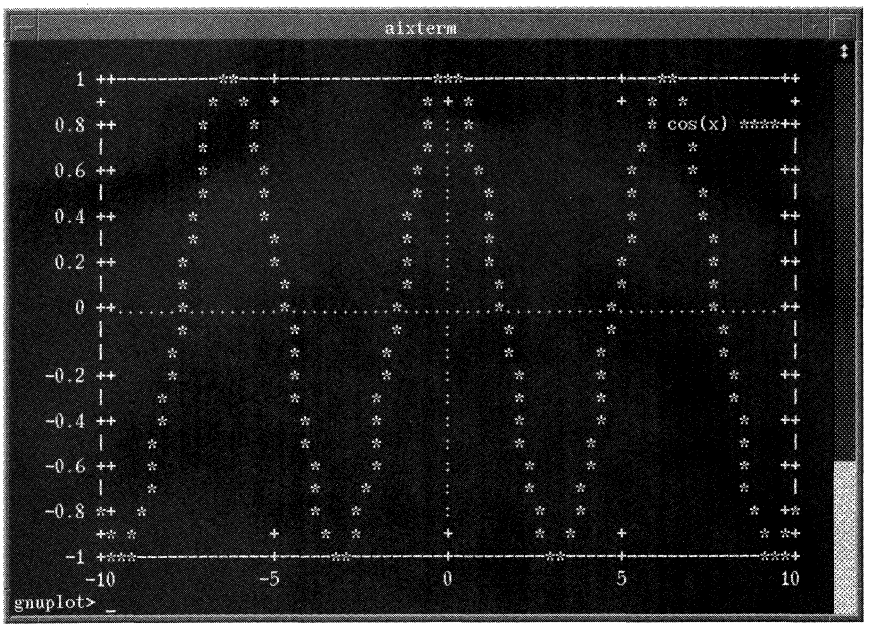

Figure 10.3 gnuplot dumb terminal.

If the output device were a Postscript device, you could use:

```
gnuplot> set term postscript
gnuplot> plot cos(x)
```

and you would get Postscript output. To see the current value of term, use the command:

```
gnuplot> show term
terminal type is dumb
```

If you set your term to be X11, you will get a much better looking graph. It may look something like that shown in Fig. 10.4.

You can define your own functions (see the plot command for more information) for any function you can dream up. For example, you can graph something like:

```
gnuplot> plot x+5*sin(x)
```

which looks like an ascending sinusoidal function.

Finally, you can set a range for the plot by prefacing the expression with a range. For example, the command:

```
gnuplot> plot [1:5] x+5
```

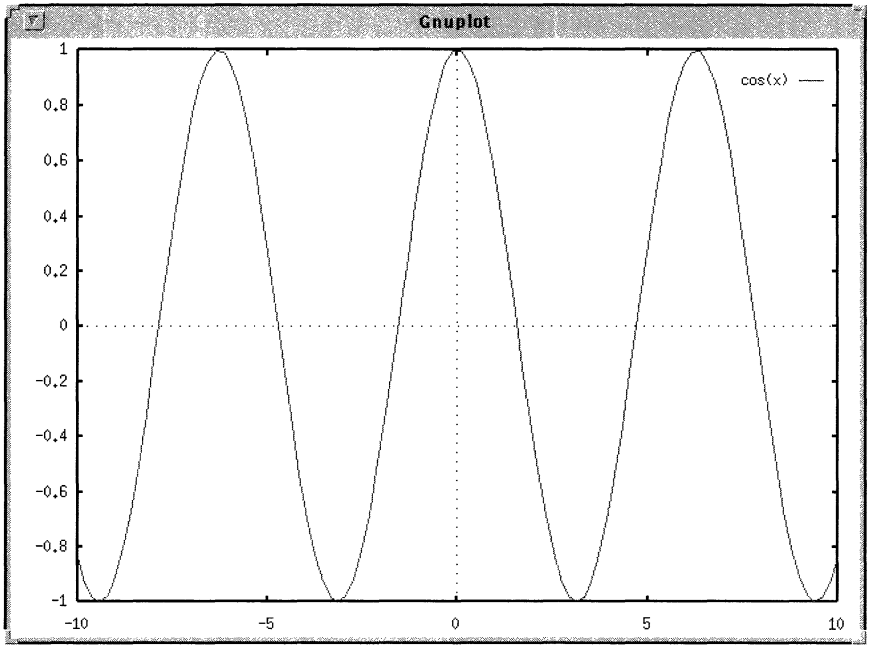

Figure 10.4 gnuplot with X11 window.

will display a straight line from 1 to 5 in the x direction and 6 to 10 in the y direction exactly as you would with a piece of graph paper. This is a simple example of range. This can be applied to any type of function.

Experiment with your own functions. There is no function you cannot display.

The other command of interest is lasergnu. This command executes the gnuplot command to produce output for an IMAGEN or Postscript printer. The basic syntax is:

```
lasergnu [-b] [-p] [-P printer] [-f file] [-t title][-help]
plot-command...
```

where -b doesn't print a banner page.
-p generates Postscript output.
-P printer spools output to printer automatically.
-f file takes plot-commands from file as input to lasergnu.
-t title gives the plot a title.
-help displays a list of options.
plot-command... is one or more plot commands.

A simple example is:

```
$ lasergnu -p "plot [1:5] x"
```

This will generate Postscript output displaying a straight line from 1 to 5 and will print to the local default printer.

10.5.3 Installation

The gnuplot follows the standard configure methodology. To create the appropriate makefiles, issue the command:

```
$ configure --prefix=/usr/local
```

You may want to examine the first few sections and make any changes you deem necessary. The only macros you will probably need to change are the destination directories for the binaries, libraries, and man pages. Change definitions such as prefix bindir and mandir to determine where you would like to place the binaries and help files. You must also set the TTERMFLAGS variable to define which devices you would like gnuplot to support. The basic list of supported devices is listed in the README file.

There is an RS/6000 makefile supplied named makefile.rs6k, if you might like to start with this file. You may still need to change destination directories for files.

You can build the appropriate devices for a given machine by simply letting the make choose the defaults for you. The build is fairly straightforward. Then you can type the command:

```
$ make
```

or

```
$ make aix32o
```

This will generate a listing of possible machine types and terminals to support. From this list chose your machine type. A good first attempt is to type a comand which builds the ALL target. This will generate the executables with certain macro definitions which gnuplot gives you when it begins the build.

There are several machines which are supported by this distribution of gnuplot. See the makefile for more information.

Note that if you are building the version for X11 support, you may have to modify the X11INCLUDE macros to specify the correct directory for both the include and library files for X11. For example, on an RS/6000 these files exist in the /usr/lpp/X11 directory. Keep this in mind if you get "can't find file" error messages. This seems to have been fixed in the newer versions of gnuplot, but keep it in mind if you see the above kind of error.

10.5.4 Conclusion

gnuplot is a very powerful plotting package which provides significant functionality to the UNIX power user. Since it provides plotting support for many devices including many newer devices, gnuplot is a very useful tool for data manipulation and display.

10.6 Tcl

10.6.1 Introduction

Tcl stands for Tool Command Language and is a scripting tool which allows both generic and X Windows applications to be rapidly developed. You can either write directly in the Tcl lanaguage or embed Tcl commands in your programs.

Tcl provides a simple to use and intuitive interface to the X Windows system as its main function. Instead of having to program to the standard X Windows libraries, you can issue simple Tcl calls to the Tcl command language interpreters to generate windows and all other X Window widgets that you normally program.

Tcl is also used as a standard scripting language which can do most functions including add, subtract, multiply, and divide, among other things. It functions as many other interpreters do and provides a series of commands for text and variable manipulation. Overall, Tcl is a tool you will want to check out to enhance your ability to develop applications rapidly.

Because of the interpretive nature of Tcl, there are performance issues which can only be addressed through a compiler. Several people, including Adam Sah and Jon Blow, have written a compiler for Tcl and have included a paper documenting their effort. Look for a Postscript file (*.ps) in the Tcl distribution and print or view this for more details.

10.6.2 Usage

The Tcl interpreter can be invoked either interactively from the command line or with a C function call from a program. You can embed a call to the Tcl parser in your program and link C functions that implement Tcl's built-in commands. This means that you can interactively enter Tcl commands to your program and make your program's execution dynamic.

Besides the embedded functions, you can invoke the interactive interpreter known as wish (WIndowed Shell). This provides a real-time, interactive Tcl command interpreter. Tcl looks most like a combination of LISP, Perl, and shell scripting.

A set of extensions known as Tk provide the actual interface to the X

Windows system. With the Tk system, you can issue Tcl commands to generate Motif-like applications with a significantly smaller number of lines of code. A good example of this is the classic Hello World example so often seen in X Windows books. Compare the code below for generating the Hello World example to those documented in several X Windows books.

```
#!/opt/gnu/bin/wish -f
#Tcl/Tk hello world script
#
button .button -text
       "Hello World"
.button configure -command "destroy ."
.button configure -foreground red -background black
.wm title . "Tcl/Tk Window"
pack .button
```

This is all that is required to generate a simple Hello World window and display it on an X Windows server screen.

The best way to learn more about Tcl is to discuss the simple example shown above.

The first line tells your shell to invoke the wish interpreter and to pass the rest of the file to wish for execution. The next two lines are comment lines which must always have a # in the first column.

The first Tcl command is the button command. The button command generates a push-button widget which is one of the simplest of the X Windows primitives. The widget is named .button. The . represents the top-level widget much like / represents the top of a filesystem. It is necessary to precede the child widget (in this case button) with . to represent a fully qualified widget. Finally, the -text option to button passes a string to the button widget for inclusion in the windows.

The next line consists of a configure command, which changes the attributes of the widget on which they are told to operate. In this case, the destroy command is used to destroy the window when the mouse button is pushed. Note that this destroy command destroys the parent widget . and, therefore, destroys all child widgets as well.

The next line changes both the foreground and background colors of the Hello World window.

The wm command tells the X Windows server software to set the title of the window to "Hello World." This could be anything you would like.

Finally, the pack command sizes the X Window and makes it visible.

Note that all of these commands can be entered interactively to the wish interpreter. In fact, this is the recommended way to debug your scripts before putting it into production.

There is a tremendous amount of documentation available in the doc

subdirectory. All files ending with a .3 are troff macro input files describing the Tcl core functions. The files ending in .n document the Tcl core commands and the Tcl.n file documents the Tcl system in general . These can be viewed with a command like:

```
$ nroff -man *.n | more
```

This will cycle you through the manual pages provided with Tcl and allow you to choose the ones to print out.

Tcl commands. Tcl commands consist of one or more fields separated by white space. The number of fields depends on the command in the first field. The first field always contains the command, while the second and third fields contain the arguments to the command. There are also a variety of variable substitutions that you can perform with brackets and braces. These are documented fully in the tcl.3 file. However, some of the more powerful commands are outlined below:

`array option arrayName arg arg ...`	Manipulates arrays.
`break`	Returns break code.
`case string`	Emulates a C language case statement.
`cd dirname`	Changes directories.
`close fileid`	Closes open files.
`concat arg...`	Concatenates all args into a single string.
`continue`	Continues to next outermost loop.
`eof fileid`	Returns end-of-file condition.
`error message`	Displays an error message.
`eval arg arg...`	Concatenates all commands and passes them to the Tcl interpreter.
`exec arg arg...`	Executes commands specified as args in a subprocess.
`exit returncode`	Exits a procedure and generates returncode.
`file option name arg arg...`	Operates on a file including time, ownership, directory name, execution, and file status.
`flush fileid`	Flushes all buffered output to fileid.
`foreach varname list body`	A looping construct.
`format formatstring arg arg...`	Generates a formatted string much like sprintf.
`gets fileid varname`	Gets the string from fileid and places it in the variable varname.
`global varname`	Declares varname as global.
`history`	Generates and controls the history list of commands.
`info option arg arg...`	Provides informations about various internals of Tcl.
`lappend varname value value...`	Appends each element specified by value to the list specified by varname.

`lindex list index`	Returns the index's element from list with 0 as the first element.
`linsert list index element ...`	Generates a new list by inserting all elements just before the index's element.
`list arg ...`	Returns all args in a list.
`llength list`	Returns length of the list.
`lrange list first last`	Returns a new list consisting of the elements first through last from list.
`lreplace list first last element...`	Returns a new list consisting of elements first through last replaced by elements specified.
`lsearch list pattern`	Returns the index of the matching string pattern in list.
`lsort list`	Sorts list.
`open filename access`	Opens filename with a specified access.
`proc name args body`	Creates a new procedure named name containing body which can then be invoked with name.
`puts fileid string`	Outputs string to fileid just as puts in C does.
`pwd`	Returns current working directory.
`read fileid`	Reads the contents of fileid.
`rename oldname newname`	Renames command named oldname to newname.
`return value`	Returns value from the current procedure.
`scan string format var var....`	Parses string and returns vars according to the format specified.
`seek fileid offset origin`	Changes the current access position for fileid.
`source filname`	Passes filname as a series of commands to the Tcl interpreter.
`split string splitchars`	Returns list created by splitting string at each character specified by splitchars.
`string option arg arg...`	Performs various string operations including compare, index, last, length, match, conversion, and a series of others.
`time command`	Times execution of command.
`trace option arg arg...`	Traces commands based on options specified.
`unset name`	Removes one or more variables.
`upvar level othervar myvar...`	One or more local variables can refer to global or enclosing procedure call variables.

The above section outlines most of the commands available with Tcl, but there is significantly more functionality available. See the man pages included with the product for more details.

10.6.3 Installation

The latest versions of Tcl and Tk are available on ftp.uu.net in the languages/tcl directory or on ftp.x.org in the contrib directory. Good luck. If you have difficulty building on AIX, there is a file called porting.notes which describes many of the isses you may encounter when building Tcl on AIX.

Using the standard configure utility, type:

```
$ ./configure --prefix=/usr/local
```

This will autoconfigure a new makefile which is consistent with your machine's configuration. You can ignore any messages unless they say ERROR. In this case, you may have to examine what error occurred and make the appropriate modifications. If Tcl isn't able to configure itself, see the Tcl.n file for more details on possible fixes.

Once you have successfuly executed the config script, you need to build the Tcl library libtcl.a with the command:

```
$ make
```

Finally, type:

```
$ make install
```

to install Tcl scripts, libraries, and binaries in the directories you configured before you invoked ./config.

If you want to ensure that you have built and configured Tcl correctly, you can type the command:

```
$ make tclTest
```

which will generate some test programs and run them against your newly built interface.

10.6.4 Conclusion

Tcl is a very powerful scripting language which provides much of the functionality of both C and shell scripts. It also provides an easy-to-use interface to develop X Windows applications. Investigate Tcl if you need to increase your productivity when developing windowed applications.

Tcl can be used as a substitute for Motif and other higher-level widget tool kits. It provides a lower level of detail but with a significantly simpler interface. The most common use of Tcl is for small applications which need a graphical user interface. Tcl may not hold up under the rigors of a large software development job and should be carefully studied before being employed in a situation like that.

If you are interested in doing X Windows programming, you will definitely want to work with Tcl.

10.7 xloadimage

10.7.1 Introduction

xloadimage is a tool which allows you to manipulate various types of files and images directly from your AIXWindows environment. You can also manipulate a variety of image types with techniques such as dithering, depth reduction, zoom, and clipping. This type of capability will significantly enhance your ability to use your windowing environment with a variety of image and file types and will increase your productivity.

10.7.2 Usage

The basic syntax for the xloadimage command is:

```
xloadimage [global_options] {[image_options] [image ...]}
```

where global_options describes a variety of options which apply to all input files.
image_options specifies options which act on specific images.
image... specifies one or more image files to act upon.

Some of the global_options are:

`-border color`	Specifies the background color of the image
`-configuration`	Displays the image path, image suffixes, and supported filters which will be used when searching for and reading images
`-default`	Uses default root to display the image
`-delay time`	Automatically displays the next image in the list after time seconds
`-display name`	Specifies name as the screen to display the image on
`-dump image_type [,option[=value]] dump_file`	Specifies a dump file of a specific type to dump the resulting image to instead of displaying it on the screen
`-fit`	Forces the image to fit the current default X Windows server color maps and visual
`-fullscreen`	Displays image on the entire screen
`-geometry WxH [{+-}X{+-}Y]`	Sets the size of the displayed image on the screen based on the standard X and Y coordinates used by X Windows
`-help [option]`	Displays help on a specified option
`-install`	Forces loading of the colormap when the window is focused
`-list`	Lists the images along the image path
`-onroot`	Displays the image on the root window
`-path`	Displays information about the program configuration
`-pixmap`	Uses the pixmap as the backing store
`-supported`	Lists the supported image types
`-type type_name`	Forces xloadimage to attempt to load the image files as a file type_name

`-verbose`	Verbose mode
`-version`	Displays xloadimage version

Some of the image_options are:

`-at X,Y`	Displays the image at X,Y coordinates.
`-background color`	Uses color as the background type.
`-brighten num`	Brightens the image by num percent, where 100 leaves the image exactly as it originally was. For example, 50 would darken the image, while 150 would lighten it.
`-center`	Centers the image on the base image loaded.
`-clip X,Y,W,H`	Clips the image, where X,Y specifies the upper, left corner of the image and W,H specifies the width and height, respectively.
`-colors num`	Sets the maximum numbers of colors of an image to num.
`-dither`	Dithers a color image to monochrome.
`-foreground color`	Specifies the foreground color of an image.
`-global`	Forces the following option to apply to all displayed images.
`-gray`	Displays an image as grayscale.
`-halftone`	Displays an image as halftone on a color display.
`-invert`	Inverts a monochrome image.
`-merge`	Merges this image onto the base image.
`-newoptions`	Resets globally specified options.
`-normalize`	Normalizes a color image.
`-rotate num`	Rotates a specified number of degrees.
`-shrink`	Shrinks an image to fit on the display.
`-smoot`	Smooths a color image.
`-tile`	Tiles an image.
`-xzoom num`	Zooms num percent in the X axis direction.
`-yzoom num`	Zooms num percent in the Y axis direction.
`-zoom num`	Zooms num percent in both X and Y directions.

The above listing does not specify all the options; however, it does list the most important ones. See the man page xloadimage.1 for more details on the specific options and their function.

Besides the options listed above, xloadimage uses an initialization file ~/.xloadimagerc. This file contains three sections, path, extension, and filter. Each section is described in some detail in the xloadimage man page. There is also an example of an initialization file in the man page, and you should reference this since it is important to proper and efficient xloadimage start-up in a particular environment.

The image dumper supports the following options. For JPEG, they are:

arithmetic

grayscale

nointerleave

optimize

quality

restart num

For PBM, they are:

normal

raw

For TIFF, they are:

compression (none, rle, g3fax,g4fax,lzw,jpeg,next,rlew,mac,thunderscan)

These options describe the types of files you can dump with the -dump option from virtually any image you can create. This is a very powerful feature which allows you to create images and save them in all the formats listed above. See the xloadimage man page for more information.

Much of the functionality available from the options described above is automatically performed by the xloadimage system. If you attempt to display images on a display which will not support the specific needs of the image, xloadimage will automatically dither the image and manipulate the color maps and visuals as necessary to give you the best image your display is capable of. xloadimage will even uncompress compressed files such as .Z and .gz (gzip) files before manipulating them for you.

Options such as -dump allow you to manipulate the image and save the newly created image file to disk instead of displaying it directly on the screen. This gives you the ability to manipulate the image with the variety of filters provided with xloadimage. Experiment with this to become more familiar with the filters provided with xloadimage.

xloadimage has a significant amount of intelligence about the type of image file it is attempting to display. It is rare that you have to specify the image type of an input file. xloadimage will sense which type of file you are passing to it and use the proper filters to manipulate it accordingly.

The man page for xloadimage gives some advise regarding the display of certain types of images. For example, most GIF and G3 fax images have aspect ratios of approximately 2 to 1. You would probably like to view it as 1 to 1 on a relatively square screen. To do this you might use the -xzoom 50 or -yzoom 200 options.

Some simple examples are:

```
$ xloadimage -onroot earth.image
```

This will load the image file earth.image onto the root window. In this case earth.image is a raster file.

The following will cause the image to be dithered into monochrome and centered on the root window. This will be much faster than the previous command on a monochrome display.

```
$ xloadimage -default -tile -dither earth.image
```

This command will double the size of the earth.image image on the screen:

```
$ xloadimage -zoom 200 earth.image
```

Finally, to dump a file like earth.image to a JPEG file, you might use a command like:

```
$ xloadimage earth.image -dump jpeg,quality=80,grayscale new.image
```

There are more examples in the man page for xloadimage.

10.7.3 Installation

There is a configuration script with xloadimage which does a good job building the proper makefile on an RS/6000. To build xloadimage, first review the makefile file for any options that may not be appropriate for your machine. The deafults should work fine with an RS/6000 running AIX in a standard configuration. Once you have reviewed makefile, type:

```
$ make clean
$ make configure
```

This will generate a new makefile which will be properly configured for your machine. Next, you need to build the xloadimage system. Use the command:

```
$ make
```

Finally, if you want to install the xloadimage binaries and associaged man pages, you can use the command:

```
$ make install
```

This will install all images in /usr/local by default. You can place them elsewhere by using the standard method as described with many other tools in this book.

There are some notes in the documentation included with xloadimage that explicitly say not to run either imake or xmkmf with it since there are too many problems caused with the AIX imake and xmkmf to be useful. The configure script included with xloadimage works just find and should be used as described above.

10.7.4 Conclusion

xloadimage it a very powerful utility which can manipulate and display a large number of image file types. By using xloadimage, you can significantly enhance your productivity in an AIXWindows environment or in any number of other X11-based windowing environments.

10.8 mpeg_play

10.8.1 Introduction

mpeg_play is known as a video software decoder. You can take MPEG input streams and display them in AIXWindows. There are a variety of standards related to MPEG, and some of them are described in the README file included with this distribution; however, suffice it to say that mpeg_play supports many of the more common and documented MPEG formats.

There are subdirectories which contain source for both the interface tool kit and the bitmaps for buttons and other display items. You may want to examine these to learn more about how the program is structured.

10.8.2 Usage

The basic syntax for mpeg_play is:

```
mpeg_play [-nob] [-nop] [-display macname] [-dither dith_option]
[-loop] [-eachstat] [-no_display] [-shmem off] [-l_range num]
[-cr_range num] [-cb_range num] [-quiet] file
```

where -nob ignores B frames.
-nop ignores P frames.
-display macname displays output on macname.
-dither dith_option selects from a variety of dithering options including:
ordered—ordered dither
ordered2—a faster ordering dither
mbordered—ordered dithering at the macroblock level
fs4—Floyd-Steinberg dithering with four error values propagated

fs2—Floyd-Steinberg dithering with two error values propagated
fs2fast—fast Floyd-Steinberg dithering with two error values propagated
hybrid—combination of ordered and Floyd-Steinberg two-error dithering
2 × 2—2 × 2 dithering technique
gray—grayscale dithering
color—24-bit full-color display
none—doesn't perform any dithering
mono—Floyd-Steinberg dithering for monochrome displays
threshold—Floy-simple dithering for monochrome displays
-loop causes mpeg_play to continually loop to the beginning of the MPEG file.
-eachstat displays statistics after each frame.
-no_display dithers but doesn't display the file.
-shmem_off turns shared memory off.
-l_range num sets the number of colors assigned to the luminance component.
-cr_range num sets the number of colors assigned to the red components of the chrominace range when dithering.
-cb_range num sets the number of colors assigned to the blue compnent of the chrominace range when dithering.
-quiet suppresses printing of frame numbers.
file is MPEG input file.

This player can display both MPEG-1 and XING input files. If you attempt to display another types of files such as multiplexed video/audio files, you may experience difficulties.

10.8.3 Installation

The installation of the mpeg_play system is very straightforward on an AIX system. Simply use the commmands:

```
$ make -f Makefile.RS6k clean
$ make -F Makefile.RS6k
```

If you want to install the resulting executables and man pages into the /usr/local area, use the commands:

```
$ make -f Makefile.RS6k install
$ make -f Makefile.RS6k install.man
```

If you are running a Gt4x card on your RS/6000, you may need to add -DRS6000 to the CFLAGS in Makefile.RS6k to ensure that the correct files are built into the final executable.

10.8.4 Conclusion

The mpeg_play executable is an extremely powerful MPEG viewer which allows you to display virtually any MPEG file on an RS/6000 as well as on a variety of other platforms. By using this tool, you can view animation in real-time and signficantly enhance the power of your AIX-Windows environment.

10.9 xearth

10.9.1 Introduction

xearth is a program which displays the earth in the root window of your UNIX workstation. It has a large number of options which include how you view it, from where, and at what time. It is kind of a neat thing to put in the root window of your workstation and does generate a lot of interest and discussion around the office. Have fun.

10.9.2 Usage

The basic syntax for the xearth command is:

```
xearth [-pos pos_spec] [-sunpos pos_spec] [-mag factor] [-size
size_spec] [-label] [-shade] [-grid] [-wait secs] [-timewarp
timewarp_factor] [-time fixed_time] [-display name] [-version]
```

where -pos pos_spec specifies the position from which the earth should be viewed. The pos_spec is a fairly complicated specification and is described fully in the man page for xearth.
-sunpos pos_spec specifies the position of the sun.
-mag factor specifies the magnification factor.
-size size_spec specifies the size of the image to be rendered.
-label provides a label at the lower right-hand corner of the window.
-shade shades the surface of the earth.
-grid displays the longitude/latitude grid on the earth.
-wait secs waits secs seconds between view updates.
-timewarp timewarp_factor scales the rate at which time passes.
-time fixed_time uses the fixed time to render the image and not the current time.
-display name is the display on which to place the xearth image.
-version displays the version of xearth running.

The above list represents just a small portion of the options available to xearth and doesn't begin to describe the options at the level of detail available in the man page for xearth. Before using the xearth applica-

tion, see its man page for more details and specifics on the xearth application.

To invoke the xearth application, simply type:

```
$ xearth
```

This will generate a root window with the earth as viewed from space at the current time from a default location. You can use any of the above options or any of those documented in the man page to modify any application behavior from there.

10.9.3 Installation

A modified makefile is included for the RS/6000. To build xearth, simply invoke make as follows:

```
$ make -f Makefile.rs6k clean
$ make -f Makefile.rs6k
```

If you want to install the generated files, use the command:

```
$ make -f Makefile.rs6k install
```

This is all there is to building xearth.

10.9.4 Conclusion

What can you say. This is a neat little program which does demonstrate some of the power of this type of program. You should examine some of the code to learn some of the tricks and techniques you can use when building a program of this type. Plus, it makes for a great conversation piece.

Chapter

11

Nonnative Communications Tools

There are several very good tools which provide communication interfaces from UNIX platforms to other platforms. This chapter presents some of the best power tools which run on most UNIX platforms.

Kermit provides file transfer and terminal emulation capabilities between most platforms today including PCs, Macs, UNIX, VMS, and others. It is widely used and is often the de facto choice when moving information between different platforms. Because of the tight budgets at Columbia and various other factors, Kermit is not distributed on the CD included with this book. However, it is very inexpensive and the documentation that comes from Columbia is much better and more complete than this section. For information on the product and how to get it, contact:

Watson Laboratory
Columbia University Academic Information Systems
612 West 115th Street
New York, New York 10025, USA, Earth
(212) 854-5126 (phone)
(212) 662-6442 (Fax)
kermit@columbia.edu

xmodem is a protocol which provides basic file transfer capabilities from one platform to another. In fact, xmodem is often supported within larger programs like Kermit and other emulation packages that you can purchase. You can, however, use xmodem by itself for file transfer if this is all you need.

tn3270 provides full-screen All Points Addressible (APA) 3270 data stream emulation. With this tool, you can make an asynchronous ASCII device look like a 3270 terminal. This is one of the most powerful

tools available and provides most of the functionality you get from very expensive 3270 emulator packages from a variety of vendors.

crttool provides true vt*xxx* emulation between platforms. While there are several packages available which provide some level of emulation, if you are using a software package which needs complete vt*xxx* terminal-style support, you may want to investigate crttool.

There are many other packages which provide communications functionality which are not mentioned here. However, these packages provide most of what you will need in these types of packages and will at least get you started in this area.

11.1 Kermit

11.1.1 Introduction

Kermit is a terminal emulation and file transfer tool which is provided free from a variety of sources, including the Internet. Its primary use is as a file transfer tool over telephone lines. It is written and maintained primarily at Columbia University by Frank da Cruz, Bill Catchings, and Jeff Damens.

There are two primary versions, of which the most popular is C-Kermit. This version is written primarily in C and is, therefore, portable to many platforms, including UNIX. The other version is called MS-Kermit and is much less commonly used and will, therefore, not be discussed.

Kermit is structured based on a client/server paradigm which consists of one machine acting as the server while the other machine is the client or remote machine. This allows for two-way file transfer using the put and get commands. You must first start the server function and then escape back to the client machine and issue commands. The server can be either local or remote and provides the maximum flexibility and power.

The primary advantage of Kermit over many other file transfer and terminal emulation package is that it is available on many platforms, including UNIX, VMS, Macintosh, DOS, OS/2, VM, and others. This allows users to use the primary interface and scripts to transfer files and information from any platform to any platform. And it is free.

11.1.2 Installation

As with all other nonnative UNIX tools in this book, the easiest way to get Kermit is from the Internet. This may mean by purchasing a tape or using a direct or indirect Internet connection to get Internet access. See Chap. 6 for more information.

There is a Kermit tar file available for a variety of versions. Check the dates and make sure you get the latest and greatest version since changes and additions are occurring constantly as the product matures. Store the kermit tar file in /tmp or some other place which will not take up permanent disk space since you will not need it once the tar file is unwound. The following will assume the tar file is called kermit.tar and the directory in which it is stored is /tmp.

C-Kermit files all begin with a ck prefix. You should create a directory for Kermit and then cd to this directory. Next you need to unwind the tar file with the command:

```
$ cd ./kermit
$ tar xvf /tmp/kermit.tar
```

Often the tar file is named ck*uxxx*.tar where *xxx* is the version of Kermit.

The files resulting from the tar command have the following naming syntax:

```
ck<system><what>.<type>
```

where <system> describes the system to which the file applies:
9: OS-9
a—documentation
c—all systems with C compilers
d—Data General
h—Harris computers
i—Commodore Amiga
m—Macintosh
o—OS/2
p—DOS PC
u—UNIX and UNIX-like systems
v—VAX/VMS
w—Wart
<what> is a mnemonic for what's in the file:
aaa—readme file
cmd—command parsing
con—connect command
deb—debug/transaction log formats
dia—modem/dialer control
fio—system-dependent file I/O
fns—protocol support functions
fn2—more protocol support functions
ker—general Kermit definitions

mai—main program
pro—protocol
scr—script command
tio—system-dependent terminal I/O and interrupt handling
usr—user interface
us2—more user interface
us3—more user interface
<type> is the file type:
c—C source
h—header files
w—Wart preprocessor source
nr—nroff/troff source
mss—scribe text source
doc—documentation
ps—Postscript documentation
hlp—help text
bld—instructions for building Kermit
bwr—bug list
upd—program update log
mak—makefile

For more information on the functionality and content of each file, see C-Kermit documentation available within the tar file. Depending on which version of Kermit you get, you may get a different groups of the files from those discussed above. See your distribution and associated documentation for more details.

A makefile is provided, as for all tools from the Internet, in an effort to assist you with the installation and building of the product. This makefile is called ckuker.mak. Rename this file to makefile and issue the make command as follows:

```
$ mv ckubs2.mak makefile
$ make xxx
```

where xxx is the platform for which you are building Kermit. Note that within the makefile there are a variety of platforms for which the UNIX version of Kermit can be built. You need to choose which platform by reviewing the makefile, which has descriptions of which platforms are available and which you should choose depending on the platform you are on. If your particular platform is not listed, choose between bsd and sys5r3, which denote Berkeley- and AT&T-derived systems, respectively. Most systems will work correctly with the SysV symbol and therefore the command:

```
$ make sysr3
```

would build a working system for you if your particular system is not specifically listed. The resulting binary will be called wermit. Test this and make sure it works; then rename wermit to kermit and install it in the bin subdirectory for general use:

```
$ mv wermit bin/kermit
```

Move the source code into the src directory and the documentation into the doc directory. This again uses the general product structure recommended for nonnative UNIX products in Sec. 6.6.

For AIX with the gcc compiler, use the command:

```
$ make clean
$ make rs6000c
```

Note that most operating systems and associated machines are supported with reasonable defaults in the makefile. See the makefile for all the different machines supported.

11.1.3 Usage

Kermit provides both interactive and command line capabilities. For short and simple file transfers, the command line capabilities will be sufficient; however, for more complex transactions, the command mode will be necessary. Each is described in detail below.

There are several things to note when using Kermit with AIX. Use SMIT and choose either the share or pdisable line characteristic. Make sure you also choose the correct baud rate. If you define the port at the same speed as the speed of the connections between the modems, you won't have to worry about flow control in the modem, and the connection will work cleaner and with more simplicity. If you don't understand how to do this, see the AIX documentation for more details.

Once you have defined the port as described above, you can use Kermit as described below. There are other ways to configure the ports, but there are ramifications with each. If you define the port as a share port, you will have difficulties using Kermit because the getty has control of the port with a login process by default. This means that the easiest way to use the port is to define it as a initilized port with no connection. If the port is in share mode, you may have to play some games to allow sharing of the port from both getty and Kermit simultaneously. If you are using Kermit alone, there is no reason to use any other default template than the initialize-only template. If you are using other subsystems such as UUCP or PPP, you may have to add an additional modem and line to support Kermit since this may be easier than manipulating the getty subsystem under AIX.

Of course, if you define the port as either penable or pdisable only, you are limited to unidirectional connections. This may be too limiting if you are attempting to establish connections in both directions.

File transfer. Kermit's primary use is as a file transfer tool over telephone lines. In local mode, stdout is continuously updated to show the progress of the file transfer. A dot is printed to the screen for every four data packets. Other packets are represented by:

I	Exchange parameter information
R	Receive initiate
S	Send initiate
F	File header
G	Generic server command
C	Remote host command
N	Negative acknowledgment
E	Fatal error
T	Time-out
Q	Damaged packet received
%	Packet retransmitted

You may control the flow and operation of Kermit during this transfer using the following commands:

CTRL-F	Stops transfer of current file and moves onto the next file
CTRL-R	Resends the current packet
CTRL-A	Displays status of the transfer
CTRL-B	Interrupts entire batch of files and terminates the transfer

Note: With System V versions of UNIX, you may have to precede the above interrupt commands with a \ to escape the control sequence. Finally, to regain control in the event of an emergency, type CTRL-C CTRL-C. This should interrupt the Kermit program and take you back to the terminal prompt. This will kill anything that was currently in progress, so do it only as a last resort.

Command line operation. The basic syntax is:

```
$ kermit [-rkxtfcniwqdh} [-s fn] [-a fn1] [-l dev] [-b speed]
[-p parity] [-g rfn] [-e len]
```

where r passively waits for files to receive.
k passively receives files and displays to stdout.
x begins server operation.
t is half-duplex with XON as handshake character.
f sends a "finish" command to the server.

c establishes a terminal connection before any protocol transaction occurs.
n is like -c but do so after a protocol transaction occurs.
i sends file without any conversions.
w is write protect mode; avoids filename duplications for incoming files.
q is quiet mode; suppresses screen update messages.
d is debug; creates debug.log file current directory with debugging information.
h displays help information.
s fn sends filename fn.
a fn1 is alternate name for file; cannot use wildcards.
l dev—dev is a terminal line such as /dev/tty1.
b speed specifies line speed.
p parity specifies parity as e,o,m,s,n.
g rfn requests a remote server to send name(d) files.
e len is extended packet length; allows for packets longer than the default 90.

Kermit uses a very standard UNIX command syntax utilizing hyphens and stdin and stdout. It also uses the client/server paradigm which allows for either of the two machines communicating to act as the server. A connection between a server and client must be established before transfers can take place. To establish a connection and a remote server, issue the following command:

```
$ kermit -l /dev/tty1 -b 2400 -n -r
```

This will establish a 2400-baud terminal connection with the remote machine where you will log in and establish the remote machine as the server. To transfer a file, enter the send command and then escape back to the local session using the CTRL-\ c command. You will be able to watch the file transfer occur. When it is finished, you will be reconnected the remote machine. You can then send another file or log off as you normally would.

The escape sequence to go from remote to local machine is:

```
CTRL-\ c
```

The CTRL-\ is equivalent to an ASCII escape. Therefore the escape sequence to return from remote to local mode is ESC-C. Because the ESC key is rarely mapped correctly, CTRL-\ is often used to simulate ESC. Note that to simulate the ESC, you must hold down both the CTRL key and \ key simultaneously. This sends a single ESC charac-

ter. Once you have done this, press the C key, and you should receive a Kermit prompt.

Local mode really means that you are logged on to a local machine or you are using an external communication line. Remote mode denotes a machine which is logged on to and is transferring files over an external line connected to your local machine. Your local machine is remote unless you explicitly point your Kermit session at an external line with a -l command.

Some other examples are:

```
$ kermit -l /dev/tty1 -b 2400 -c | vt100
```

Note that kermit provides generic terminal emulation, and you can, if you wish, pipe this through programs which interpret terminal sequences and control access to the screen. A common one is a vt100 program, which allows for DEC vt100 terminal emulation:

```
$ kermit -l /dev/tty1 -nf
```

This will shut down the remote server and establish a terminal session to the remote machine. This is particularly useful if you with to stop file transfer activities and resume interactive use of the remote machine. Note that this works because of the use of the n option which allows the f option protocol to finish before the n option begins. If you were to use -cf, this would not work correctly.

This command will transfer all remote files with the name kevin.* to your local directory:

```
$ kermit -l /dev/tty1 -b 9600 -qg kevin.\* &
```

There are several things to note about this command. Kermit will be invoked in the background due to the &. This allows work to continue while the transfer is occurring. The q causes Kermit to suppress any output to the terminal, so you don't get those annoying messages to the screen during transfer and usage. Note also that Kermit understands wildcards for filenames; however, the only ones that are recognized and handled correctly are * and ?. The * will do a multicharacter substitution while the ? will do a single-character substitution. All other shell metacharacters such as [and { are ignored. Finally, note that we had to escape the * with a backslash so that the shell didn't interpret it before passing it to Kermit.

Kermit uses standard input and output, and you can therefore use pipes and redirections with Kermit to provide more functionality in the tradition of UNIX.

Terminal emulation and modems. The command:

```
$ kermit -l /dev/tty1 -b 9600 -c
```

will provide generic terminal emulation and allow you to log on to a remote machine over a serial line. This will work over any kind of serial line, local as well as remote. If you are using a local connection such as a null modem cable between two RS-232 ports, this will connect you directly to the getty subsystem on the remote machine and you will get a login prompt. If you are using a switched line with modems, this command will give you control of the modem. Once you have control of the modem, you can issue standard modem commands such as those for Hayes-compatible modems. Assuming you have a phone line with two compatible modems and the remote is set up for autoanswer, you can issue the following commands to log in over the phone line to the remote machine:

```
$ kermit -l /dev/tty1 -b 9600
```

At this point, you are talking directly to the modem, which has its own command language and syntax. Most modems are Hayes compatible and follow a standardized command syntax. If you have a Hayes-compatible modem and want to now connect to a remote system, issue the following command:

```
ATDTnumber
```

where number is the phone number you wish to dial. For example, 5554404 would be a valid dial string, so the command would read:

```
ATDT5554404
```

This works exactly as you would dial using a phone. For a long distance call, you would again follow the same structure as the normal phone. For example:

```
ATDT13125554404
```

At this point, the remote modem and your local modem will begin a handshaking sequence to establish baud rates, error correction protocols, and compression techniques. There are many Hayes commands available which control virtually every aspect of your modem. You should read your modem documentation for more information. To print out the configuration of your modem type:

```
AT&V
```

At this point, you will connect to the remote machine and should get a login prompt. Log on as you normally would and proceed as if logged on to a LAN. You now have generic terminal emulation and file transfer capabilities. If you are interested in using the machine in an interactive mode as if you were logged on locally, you can proceed; however, if you would like to transfer files, you must first put the remote machine in "server" mode and escape back to the local machine with the proper control sequence (probably CTRL\-C). You issue the receive command, and your local machine waits for any files to arrive. When the transmission is finished, you are returned to your remote session where you can log off or continue with your work.

An example of this would be as follows:

```
login: kevin
password:xxxxx
Welcome to foobar, AIX 3.2
$ kermit -x
CTRL\-C

$ kermit
kermit> receive remote-file local-file

$ logoff*(Note we are back at foobar now, so we need to log off.)*
$*(We're back at the local host.)*
```

remote-file from the remote machine is copied to local-file on your local machine. Note that paths can be fully qualified or relative in the standard UNIX way. There are commands to change directories and much more. These are typically available from within Kermit in what is known as interactive mode. This is the most powerful way to use Kermit.

Interactive mode. Interactive mode in Kermit is the preferred way to perform anything but the simplest of tasks. It provides a simple interface to the entire functionality of Kermit. Through the use of simple commands, you can perform logins and file transfers as well as run scripts, redirect files, and perform modem control.

The Kermit prompt is C-Kermit>. At this point, you can type any valid Kermit command and even issue standard UNIX commands. When you invoke Kermit, it first looks in your home directory and then your current directory for a file named .kermrc. This must contain Kermit commands only and cannot contain UNIX commands. An example will appear later in this chapter.

A brief list of Kermit commands is as follows:

%	Comment
= !	Executes a UNIX command from within Kermit

`bye`	Ends session and logs out remote server
`close`	Closes a log file
`connect`	Establishes a session to either a modem or remote machine
`cwd`	Changes working directory
`dial`	Dials a telephone number
`directory`	Displays a directory listing
`echo`	Displays arguments literally
`exit`	Exits from program
`finish`	Exits remote server but doesn't log off
`get`	Gets file(s) from remote server
`hangup`	Hangs up the phone
`help`	Displays help for commands
`log`	Opens a log file
`quit`	Same as exit
`receive`	Receives files from remote server
`remote`	Issues command to remote server
`script`	Executes a login script with remote system
`send`	Sends file(s)
`server`	Begins server operation
`set`	Sets various parameters
`show`	Displays set parameters
`space`	Displays current disk space usage
`statistics`	Displays statistics about most recent transaction
`take`	Executes commands from a file

These commands are the first entered on the Kermit interactive command line. The Kermit command interpreter is relatively sophisticated in that it accepts unique shorthand commands for the above commands. You can also use the ? to prompt for proper responses. This is a very powerful feature that allows Kermit to help you finish commands. For example, if you type set and can't remember which set option you would like to use, you can type:

```
C-Kermit> set ?
```

and Kermit will provide you with a list of possible options from which to choose. This is available for all Kermit commands and is quite helpful. The other useful command is help. If you type help, you will be presented with a listing of all Kermit commands. If you type help command where command is one of the available Kermit commands, you will get additional information on that particular command itself. The other useful feature of the interactive interface is the ESC capability. If you need to fill in the rest of a keyword or request a default value, simply press ESC (again, this may be CTRL\) and Kermit will fill in the

blanks. This is very useful since you will often forget defaults and keywords.

A brief description of each kermit command follows.

%. This is useful for inserting comments into script files.

! [command]. This allows you to execute shell commands from within the Kermit interpreter. For example, to determine whether a file is in your current directory or not, you could issue the following command:

```
C-Kermit> ! ls
```

This will list all files in the current directory. Note that you can issue any UNIX command; however, you again must be careful of the symbol substitution. Be sure to separate the ! and the command with at least one space. If you want to fork a new interactive shell, issue the ! without any command argument. To exit from this shell, type exit or CTRL-D.

bye. This stops the remote screen and kills the connection between your local machine and the remote one. This will return you to your local machine.

close. This closes a log file created by the log command.

connect. This command establishes a session with either the locally attached modem or the remotely attached machine. This works in coordination with the set line command to establish the connection. This is the best way to establish a direct connection to the modem and have access to the Hayes command set directly on the modem. Examples are given below. To get back to the local Kermit prompt after issuing the connect command, you must issue the ESC as documented earlier. For example, CTRL\-C will bring you back to the Kermit prompt where you can issue any commands you normally would. To reconnect, issue the connect command again.

cwd [dir-name]. This allows you to change your local current working directory to dir-name.

dial [tele-string]. This command issues the dial command through the local modem. You can bypass any direct interaction with the modem and Hayes command structure as described earlier with this command. For example:

```
C-Kermit> dial 13125554404
```

will connect to the modem and issue the command ATDT13125554404 just as documented earlier. This has the disadvantage that you don't have control over other parameters of the modem, and you may have to change other properties of the modem. This must be done from the Hayes direct modem connection with the connect command. This command is used in coordination with the set line and set modem commands as documented below. Note that you will get a "connected" message when the modems have established a link; however, you will then be placed back in interactive Kermit mode. You then need to type connect to tie in to the established link. For example:

```
C-Kermit> dial 13125554404
connected....
C-Kermit> connect
login:....etc.
```

directory [dirname]. This displays a file listing of the current working directory or of the dirname directory.

echo [text]. This provides access to the terminal screen. You would typically use this within a script file to issue messages to the screen.

exit. This exits from the Kermit program and logs off from the remote machine and associated processes. This will place you back on your local machine in the same state as before you issued the first Kermit command.

finish. Same as bye.

get filename [filename1]. This sends a request to the remote Kermit server to send the file filename. This requires that the remote Kermit server be started and a connection established. You can optionally rename filename to filename1 on the receiving machine. An example of using a modem and getting a remote file named /tmp/foobar and placing it on the local machine as /tmp/kevin follows:

```
$ kermit
C-Kermit> set modem hayes
C-Kermit> set line /dev/cua0
C-Kermit> set speed 9600
C-Kermit> dial 9,13125554404
connected...
C-Kermit> connect
login: kevin
passwd: xxxxx
Welcome to SunOS 4.2, Have a Good Time
```

```
$ kermit
C-Kermit> server
CTRL\-C*(to escape back to local machine)*
C-Kermit> get /tmp/foobar /tmp/kevin* gets the remote file
/tmp/foobar and places it on the local machine as /tmp/kevin*
C-Kermit> exit
$
```

You are now back at the local machine with the $ prompt.

hangup. This command hangs up the local modem and kills the connection.

help [command]. This provides help on both what commands are available and more specific help on a particular command.

log {packets, session, transactions, debugging} filename. This establishes a log file of various aspects of Kermit's operation. The packets option allows for a trace of all packets in and out of the communications port. The session option provides for screen trapping. The transactions option keeps a record of all files transferred. Finally, the debugging option provides information on the internal workings and operation of Kermit and is probably only useful to developers and real hacks. You issue the close command to close and save this log file.

quit. This is the same as exit.

receive [filename]. This passively waits for the receipt of a file. Note that this is very different from the get command, which actively gets a file from a remote machine; receive waits for the file to arrive. This means that the remote machine must issue a send before the receive will process any incoming information. If we wanted to send a file to a remote machine, we could use receive as follows:

```
$ kermit
C-Kermit> set modem hayes
C-Kermit> set line /dev/cua0
C-Kermit> set speed 9600
C-Kermit> dial 9,13125554404*(note we are using the standard hayes
dialer string) connected...*
C-Kermit> connect
login: kevin
passwd: xxxxx
Welcome to AIX 3.2, Have a Good Time
$ kermit
C-Kermit> receive /tmp/kevin
CTRL\-C*(to escape back to local machine)*
C-Kermit> send /tmp/foobar
C-Kermit> exit
$
```

This sends the local file /tmp/foobar to the remote machine as /tmp/kevin. Note that we could have typed connect at the local C-Kermit> prompt and reconnected to the terminal session on the remote machine if we wanted to.

remote. This allows you to issue commands to the remote machine. There are several remote commands available:

`remote cwd [directory]`	Changes remote working directory
`remote delete filename`	Deletes remote filename
`remote directory [dirname]`	Lists files in remote dirname or current directory
`remote host command`	Executes command on remote machine
`remote space`	Displays remote disk capacity and usage
`remote type [filename]`	Displays remote filename on screen
`remote who [user]`	Displays remote user(s)
`remote help`	Displays remote server's capabilities

script. This allows for execution of canned scripts using send and respond strings. See Kermit documentation for more information.

send filename [filename1]. This sends the file named filename and, optionally, renames it filename1 on the remote system. Note that the other machine must be either in server mode or have issued the receive command. See the example above.

server. This places Kermit in server mode. All subsequent commands, such as send, the remote commands and finish, must come from the other Kermit machine. See the above example.

set [variable value]. This allows you to control virtually every aspect of the communication between Kermit machines. There are many options and not all will be documented here, only the most important. See the Kermit documentation for more information.

`duplex {full,half}`	Determines full (two-way) or half (one-way) communication.
`file type {binary,text}`	Determines whether or not translation takes place when file is transferred. If the file contains text only, use set file type text, and any necessary file translations will take place; use binary to turn off translation. See example below.
`flow-control {none, xon/xoff}`	Controls flow control based on end-to-end connection.
`incomplete {discard,keep}`	Keeps or discards incomplete file if line goes down.

`line [device]`	Used to establish line to use for communication.
`modem-dialer {direct,hayes, racalvadic,...}`	Used to establish type of modem or direct connection. Note this must be used *before* the set line command.
`parity {even,odd,mark,space,none}`	Establishes parity.
`speed {0,300,1200,2400,9600,19200}`	Establishes line speed. Use *after* the set line command.

the set file type command is extremely useful when moving between diverse machines. If you are using an ASCII machine and transferring to or from a non-ASCII machine, Kermit will perform any translations necessary to make the file readable on the other machine. For example, sending a text file from a UNIX machine to an IBM mainframe in text mode will automatically translate the ASCII characters to EBCDIC. Do not use the text mode if the file contains any nontext information. For example, tar files contain nontext information and you, therefore, must use the set file type binary command, and the remote machine must understand the tar format. This is the same for all nontext format files.

To display the values of these variables, use the show command.

show [variables]. This is used to display the value of a variable established with set.

space. This displays information on local disk space and usage.

statistics. This displays statistics about the most recent Kermit transaction.

take [filename]. This runs an file containing Kermit commands. Note that this cannot send any information after the connect is issued. For this, you should use the script command. A typical use of the take command is to establish a connection and get a login prompt from the remote machine. For example:

```
set modem hayes
set line /dev/tty1
set speed 9600
dial 13125554404
connect
```

From here you must log on interactively. Remember that when you enter Kermit, it does a take on the file .kermrc, and this is where you would typically place commands like those above. If the above file were named take.file, you would execute it as follows:

```
C-Kermit> take take.file
```

You would see the connected message and proceed as you normally would.

11.1.4 Conclusion

Kermit is a very powerful tool which provides generic terminal emulation and file transfer capabilities over serial lines. It runs on virtually every platform and provides sufficient granularity of control that you can do almost anything you would like to do when transferring files between diverse machines. Transfers between mainframes, workstations, Macs, and PCs work without a hitch. One of the gotchas about UNIX is that there is a newline character at the end of every line instead of the more standard CR-LF. When transferring files between machines, Kermit takes care of this for you. Also, when moving from an ASCII to EBCDIC machine such as an IBM mainframe, Kermit will perform the translation automatically.

For more information, print out the documentation provided with Kermit; it is very good and includes more information than is here.

11.2 xmodem

11.2.1 Introduction

xmodem is a commonly used protocol and comes shipped with most UNIX machines and PC and Mac terminal emulators. It is used by tools like Kermit to transfer files and move information from one platform to another.

xmodem is based on a client/server model where you define one end of a communication link as the server and the other end as the client. User interaction with xmodem is very similar to that used by Kermit.

xmodem is in fact a protocol and as such handles data transmission errors and automatically generates retransmission of bad packets. This allows the higher-level applications such as Kermit and the xmodem application itself to assume that all packets received are correct and to proceed accordingly.

The latest version of xmodem (3.4) is probably the last if not one of the last versions of xmodem to be produced since most work on xmodem has stopped. While a good tool, xmodem has been superseded by tools like zmodem and Kermit for file transfer. This distribution does seem to work well, however, and is of value to anyone who wants xmodem support and capabilities.

11.2.2 Usage

The basic syntax for the xmodem command is:

```
xmodem [rb|rt|sb|st|] [ymkdlx] [file...]
```

where rb receives binary.
rt receives text.
sb sends binary.
st sends text.
y uses ymodem protocol.
m uses MODEM7 batch protocol.
k uses XMODEM-1K file transfer protocol.
d deletes the xmodem.log file before file transfer is begun.
l doesn't create the log file.
x is debug mode.
file... is one or more files to be transferred.

One machine must be set to send a file while the other must be set to receive a file. To interrupt any file transfer activity use C-x C-x.

A simple example follows. To receive a file named kevin.text, use:

```
$ xmodem rt kevin.text
```

On the serving machine, type:

```
$ xmodem st kevin.text
```

This will transfer the file across the link. Note that you can send multiple files with the YMODEM protocol. A simple example follows. On the receiving machine, use:

```
$ xmodem rty file1 file2 file3
```

On the sending machine, type:

```
$ xmodem sty file1 file2 file3
```

Three files will be transferred. See the man page for more information.

11.2.3 Installation

This software comes in three shar archives plus a patch for the part01archive. Simply uncompress the archives (if necessary) and remove all comments up to the first shell command (in this case each begins with an echo command). Once you have removed the preceding comments, issue the command:

```
$ sh partxx
```

where xx is 01, 02, or 03.

Finally, you need to shar the patch1 archive with the command:

```
$ sh patch1
```

This will update the files in the current directory.

Note that xmodem is built for the Berkeley system and as such assumes certain include file structures. To properly build xmodem, you need to use the Berkeley include files. However, due to certain other problems, you must also make some changes to support some aspects of the System V model. These changes are described in a section of the README file which discusses the System V porting issues. What you must do is:

1. Change includes in xmodem.h from <sys/time.h> to <time.h> and from <sgtty.h> to <termio.h>
2. Change the string rindex to strrchr in batch.c
3. Change getput.o to getput.sysv.o in the first line of the makefile

Then you can build xmodem for AIX with the make command as follows:

```
$ make CC=gcc CFLAGS="-O -traditional -I/usr/include"
```

This is all there is to building xmodem.

11.2.4 Examples

The best way to understand how to use the xmodem command is to view a few simple examples.

Sending a file to a remote machine. To send a file to a remote machine, first log on to the remote machine with any standard terminal emulation program such as Kermit. Move to the directory to which you want the file transferred. On the remote system, type:

```
$ xmodem -r testfile
```

This starts the xmodem protocol for receiving on the remote machine. Now you must interrupt the terminal emulation session with the proper command. For example, with Kermit you would type C-c while when using Asynchronous Terminal Emulation (ATE), you would type C-v. Each different terminal emulator has some command sequence to interrupt your session and place you back on the local machine. See your documentation for more details on your specific implementation.

Once you are back at your local prompt, you need to initiate a send procedure from your terminal emulation system. For example, with Kermit you would issue a send command from within Kermit; from

many emulation programs, you can initiate a send from a pull down menu or from within a terminal emulation program with a command like send. For example, with ATE you would issue the command:

```
s testfile
```

Each implementation is different. See your documentation for details.

Note that the emulation program must support the xmodem protocol. It will say so explicitly if it does. File transfer occurs and is documented in terms of packets sent and error rates. Once the file transfer occurs, you are returned to the interactive prompt on the local machine. You can then reconnect and log off or do whatever you would like on the remote system.

Receiving from a remote machine. Log on to the remote machine as described above and issue the command:

```
$ xmodem -s testfile
```

Return to the local machine and issue a receive command. This will receive the remote file testfile from the remote machine. Keep in mind that you can rename the file on the local machine for the receive if you would like.

Once the transfer is finished, return to the remote machine and log out or perform any activities as you normally would.

11.2.5 Conclusion

xmodem is not only a protocol supported by most emulation and file transfer programs but is an application as well. xmodem is available for most platforms but may not be shipped by the vendor with their implementation of UNIX. If it is not, see the implementation on the CD that accompanies this book.

11.3 zmodem

11.3.1 Introduction

zmodem is first and foremost a file transfer protocol which allows file transfer to occur over serial lines (local and dial-up.) It superseded xmodem and ymodem and their associated protocols. While xmodem accomplishes relatively simple file transfers and provides some basic integrity checks, zmodem goes beyond this to provide multifile transfer capabilities as well as a higher level of redundancy checking to ensure

file integrity. zmodem is the choice of most PC and UNIX people (perhaps the Kermit people would object) for file transfer.

11.3.2 Installation

The zmodem tar file for the current version (2.6) is simple to unwind with the basic tar command. Once you have done this, you need to compile the two executables related to send and receive. There is a simple Makefile which specifies System V, Xenix, 386 Xenix, or Berkeley 4.*x*. If you simply type make, you will see what systems are directly supported. For example:

```
$make

Please study the #ifdef's in rbsb.c, rz.c and sz.c,
then type 'make system' where system is one of:
       sysv            SYSTEM 5 Unix
       xenix           System 3/5 Xenix
       x386            386 Xenix
       bsd             Berkely 4.x BSD, and Ultrix
```

The implementation of System V seems to be relative to Version 2, which is somewhat old and out of date; however, it may be useful for a system based on a newer version of System V.

As an example, on an RS/6000, zmodem built using:

```
$ make bsd CC=gcc
```

The other thing to note is that the Makefile is very simple and is hardcoded to use CC. gcc seemed to work better, so it was necessary to modify the Makefile to use gcc instead of CC. See the Makefile for details.

This is all there is to building zmodem. See the Makefile for more information. There is also a way to interactively get code; see the README file for more information.

11.3.3 Usage

There are two basic executables (rz and sz) and four associated symbolic links to those files (rx, rb, sx, and sb.) The r executables refer to receiving a file or files, while the s executables refer to sending one or more files. The z refers to the zmodem protocol, the x to the xmodem protocol, and the b to batch sending of files using either xmodem or ymodem.

The basic syntax for the s commands is:

```
sz [-+abdefkLlNnopqTtuvyY] file ...
sb [-adfkqtuv] file...
sx [-akqtuv] file
```

```
sz [-oqtv] -c command
sz [-oqtv] -i command
```

where - denotes standard input.
+ appends transmit data to an existing file.
a converts newline characters to CR/LF for UNIX-to-PC file transfer.
b is binary mode.
c command sends command to receiver for remote execution; returns when command execution is complete.
d changes all . to / in the transmitted filename.
e escapes all control characters.
f sends full pathname.
i command sends command to receiver for execution; returns when command is received.
k sends files using 1K blocks instead of the default 128-byte blocks.
L N uses ZMODEM packets of N length.
l N pauses for confirmation of data receipt from receiver every N bytes.
n sends file if destination file does not exist; overwrites desination file if source file is newer than the desination file.
N sends file if destination file does not exist; overwrites destination file if source file is newer or longer than the destination file.
o disables 32-bit cyclical redundancy check (CRC).
p does not tranfer file if destination file already exists.
q is quiet mode.
r resumes interrupted file tranfer.
t tim changes timeout to tim tenths of a second.
u unlinks file after transmission.
v is verbose mode.
y overwrites any preexisting file.
Y overwrites any preexisting files and skips any files which do not have the file with the same pathname on the destination system.
file ... is one or more files to transfer.

The basic zmodem commands work just as the Kermit commands do. You must first set up one end of a connection as a sender and then move back to the other system and use the receive function.

A simple example uses Procomm to receive a file. First log on to the UNIX machine with Procomm terminal emulation and then type:

```
$ sz *.c
```

Go back to Procomm and tell it to receive a file using the zmodem protocol. This will transfer all files ending with a .c suffix in the current directory back to your PC.

The receiving end of zmodem works much the same way as the send side. The basic syntax is:

```
rz [-+abepqtuv]
rb [-+abqtuv]
rz [-1abceqtuv] file
[-] [V] rzCOMMAND
```

where - denotes using standard input.
+ appends transmit to an existing file.
a converts newline characters to CR/LF for UNIX-to-PC file transfer.
b is binary mode.
c requests 16-bit CRC.
e escapes all control characters.
p does not tranfer file if destination file already exists.
q is quiet mode.
t tim changes timeout to tim tenths of a second.
u unlinks file after transmission.
v is verbose mode.
file is command file to append transferred information to.

The r commands are the receive-style commands which allow you to receive a file or files from sender.

The rzCOMMAND allows you to take input to the receiver and execute it as a command file. This is typically used for mail and other remote execution process needs.

zmodem supports a concept called AutoDownload, which means that the other end of a connection is automatically started when it receives the appropriate zmodem command. This is the case with a tool like ProComm; it automatically begins a file tranfer when the opposite end (typically on a UNIX machine) is invoked with a zmodem command.

There are several strange behaviors of this package with different types of machines, but the one to watch out for with UNIX machines is that zmodem tends to have problems when used in conjunction with cu. Both attempt to take characters from the input stream, and this can cause problems. You should use some other communications mechanism when using zmodem with UNIX.

Much of this syntax was taken directly from the included manual pages. Just look for *.1 in the directory and print or display them for futher information.

11.3.4 Conclusion

zmodem is a fairly sophisticated protocol which allows you to transfer one or more files to a remote machine over a serial line. While zmodem is both a tool and a protocol, it is included in most terminal emulation and file transfer packages for PCs and other types of workstations. This means that you can use this to transfer files to and from your UNIX machine with a variety of different tools on the other end of the link.

Chapter

12

Games

12.1 Introduction

No good book on free software would be complete without a chapter on games. The CD included with this book includes several well-known games that are available from the Internet. However, it certainly doesn't do justice to the large collection of good games available from the Internet.

The games have not been prebuilt, but the full distribution is available with each game included on the CD. A short description of each game is included in this chapter. You will need to build them and proceed at your own risk. Good luck and have fun.

12.2 Games Overview

Here are the games available on the accompanying CD:

gnuchess	The GNU chess game; pretty cool
xboard	An XII-based interface to gnuchess
gnushogi	A game based on Shogi (Japanese chess)
xshogi	The X-based interface to gnushogi
gnugo	GNU's version of Go
xtrek	X-based space war game
cbzone	Network-based tank game
xconq	X-based strategy game; considered by some to be the best multiplayer strategy game available today
nethack	Visual adventure game similar to Rogue

Chapter

13

Nonnative Internet Tools

With all of the work going on in the world of the Internet, no developer's book of tools would be complete without at least some of the most widely used tools available for the Internet. Tools such as Archie, Gopher, and Mosaic are becoming the hottest pieces of software available for UNIX platforms. Of course, the assumption is that you have a full IP Internet connection before you can use these tools, and this book won't get into the details of this connection; however, there is some discussion of this link in Chap. 6 as well as a listing of some possible Internet access providers in App. D. Please see these two areas for more information.

Again, the assumption in this chapter is that you have a full IP-level connection to the Internet and that the speed is fairly high. All of these tools will work on a lower-speed link; however, the performance will be significantly impaired. Keep this in mind as you begin to use the tools described in this chapter. Good luck.

13.1 Archie

13.1.1 Introduction

Archie is a tool which provides a search capability for the Internet. With Archie you can search for specific strings in index databases around the world. A more complete discussion of Archie and its capabilities is contained in Chap. 6. See this for more information. This section will focus exclusively on Archie and its capabilities on UNIX and, more specifically, AIX.

One thing to note about this Archie client software is that it is based on the Prospero protocol. This protocol offloads a significant amount of work from the Archie servers and provides significantly better Archie

query performance. The software included with this book is a subset of the full Prospero archie client. See the accompanying documentation for more details on the full archie client software.

13.1.2 Usage

The basic syntax for Archie is:

```
archie [-cers] [-a] [-l] [-t] [-m hits] [-N [level]] [-h hostname]
[-o filename] [-L] [-V] [-v] string
```

where -c searches substrings and is case sensitive.
-e provides exact string match.
-r searches using regular expression.
-s searches substrings and is case insensitive.
-a places results in Alex filenames.
-l places results in parsing format.
-t sorts the results by date.
-mhits specifies the maximum number of matches.
-N [level] sets the "niceness" of the query. This controls its wait for resource.
-h hostname specifies the appropriate Archie server to query.
-o filename puts results into the file named filename.
-L lists the known Archie servers.
-V is verbose option.
string specifies a string to search for.

The best way to understand how to use Archie is through examples. A simple example of a search using Archie is:

```
$ archie emacs
```

This will return all matches (up to 95) found on the default configured Archie server. If you find that this is slow, you may try a different archie server. For example:

```
$ archie -L
$ archie -h archie.unl.edu emacs
```

These commands will first list the available Archie servers and then allow you to pick one (in this case archie.unl.edu) to query for the string emacs. Your default Archie server is established when you build Archie (see next section); however, you can dynamically change your Archie server by setting the environmental variable ARCHIE_HOST. Set this to the primary Archie server for your session.

You can also use regular expressions in your search. For example:

```
$ archie '[xX] [lL]isp'
```

This will search for any combination of x and l followed by a isp string including X and L. Note that you need to quote the regular expression to modify the shell's interpretation of it. You can also use the - to denote any search strings. For example:

```
$ archie -s - -string
```

denotes that you are going to perform a case-insensitive search for the string -string. The use of a single hyphen (-) denotes that all text following the hyphen is the string to be searched for.

It is recommended that you use the nice capability of the Archie system to prioritize your Archie queries. The nice priorities are similar to those of UNIX nice and range from 0 (normal) to extremely nice (10000) and nicest (32765). Choose the higher nice numbers to lower your priority at the Archie server. For those jobs which are large or for which you don't need quick response, use a higher nice number.

13.1.3 Installation

The installation of Archie is very straightforward for AIX. The README file contains general notes; however, there is a file already configured named Makefile.rs6k. Use this makefile to build archie. Use the commands:

```
$ make -f Makefile.RS6k clean
$ make -f Makefile.RS6k
```

This will generate the Archie client system. To install this, use the command:

```
$ make -f Makefile.RS6k install
```

That is all there is to it.

13.1.4 Conclusion

Archie is a very powerful tool for queries of anonymous FTP sites around the world. The Prospero client included with this book is only one section of a much bigger software system which allows for much more sophisticated functionality than Archie does. See its accompanying documentation for more information on sites and other software capabilities of Prospero.

13.2 Xarchie

13.2.1 Introduction

Xarchie is an X11 client interface to the Archie system. It is very similar in function to Archie as described above; however, it has a more sophisticated user interface based on the X11 GUI. Chap. 6 contains a more detailed discussion of Archie. The purpose of this chapter is to give you a basic understanding of the Xarchie system and to provide you with enough information to proceed with its use.

Just as with Archie as described above, Xarchie uses the Prospero protocol to improve the performance of the Archie server. See its accompanying documentation for more information.

13.2.2 Usage

As mentioned above, Xarchie is an X11-based interface to the Archie server system. Most interaction can be done with a mouse, requiring very little knowledge of the underlying operation of Xarchie itself. The basic Xarchie screen is shown in Fig. 13.1. As you can see, this screen is quite straightforward and self-explanatory.

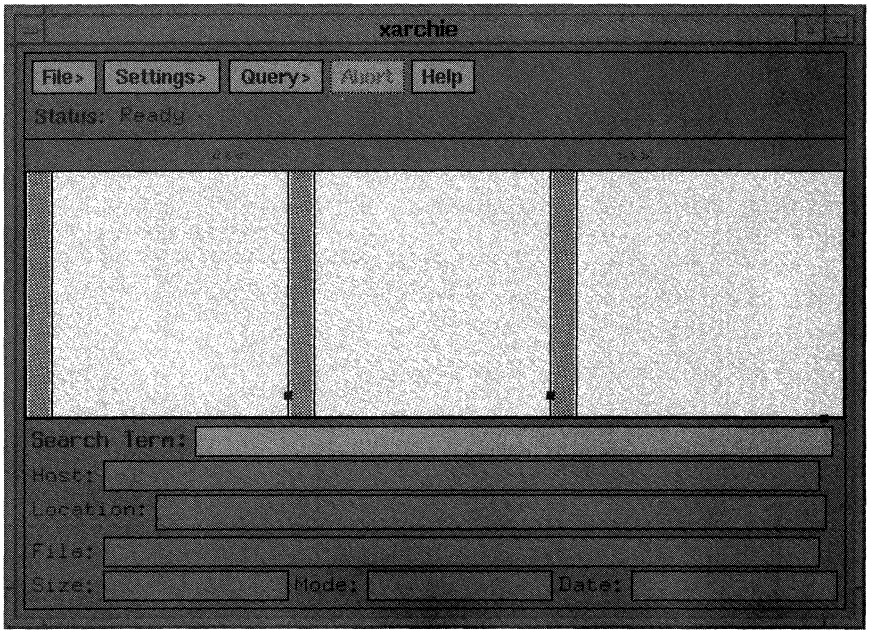

Figure 13.1 Xarchie screen.

You invoke Xarchie from the command line as follows:

```
xarchie [X Toolkit Options] [-host hostname] [-search
type|-e|-c|-s|-r|-ec|-es|-er] [-sort type|-t|-w] [-maxhits num]
[-offset num] [-nice lev|-N lev] [-noscroll]
[-mono|-gray|-color] [-debug num|-D num] [-help|-?]
```

where X Toolkit Options can contain any number of X11 resource definitions. Many of these are documented in the Xarchie man page beginning on p. 14. Things such as browser, font, and query resources can be controlled from either the command line or the .Xdefaults file. See the Xarchie man page for more details.

-host hostname specifies the Archie server name.
-e searches for exact match.
-c searches for substring.
-s searches for substring.
-r searches for regular expression.
-ec searches for subcase; exact match.
-es searches for substring; provides exact match.
-er searches for regular expression; provides exact match.
-sort type sets the sort mode for displaying archie responses.
-t sorts based on date.
-w sorts based on weight.
-maxhits num sets the maximum number of matches.
-offset num sets the offset of the Prospero query.
-nice level | -N level sets the nice level.
-noscroll tells Xarchie not to scroll automatically.
-mono | -gray | -color determines the visual type of the display.
-deug level | -D level sets the debug level.
-help | -? displays help.

This is by no means an exhaustive listing or discussion of Xarchie; however, much of the interaction with Xarchie occurs with a mouse and not with a command line. Therefore, there is much that does not need to be documented. There are a tremendous number of nonwidget and widget resources that can be set which will dramatically affect the behavior of Xarchie. See the Xarchie man page for a discussion of all the possibilities.

13.2.3 Installation

The installation of Xarchie can be quite complex; however, in this case there is a relatively straightforward configure which should work. You can attempt to build Xarchie using the commands:

```
$./configure --prefix=/usr/local
$ /usr/bin/XII/xmkmf
$ make Makefiles
$ make
```

This first generates a new makefile template and then generates a new Makefile which is appropriate for the current configuration. If you want to install the appropriate files, use the command:

```
$ make install
```

Note that if you get s SIGSEGV violation and a corresponding core dump, you may have to recompile with the -D_NONSTD_TYPES flag.

If you encounter difficulties with this method, you may have to manipulate your imake and xmkmf software. See Sec. 7.17 for more information.

There are a couple of things to watch out for in this file. See the INSTALL.RS6k and INSTALL files for more details. There is also a file named PROBLEMS which contains a variety of information if you have problems with this process. For example, if you have problems with an unresolved reference XtStrings when you link the program, it means that your X11 subsystem hasn't been completely installed. You need to modify either the Makefile or the Imakefile and rebuild the product to avoid this linking problem. This and other problems are documented in the PROBLEMS file.

The Imakefile needed to be modified to support building xarchie. See the Imakefile and Imakefile.orig files for differences.

13.2.4 Conclusion

Xarchie provides all of the functionality of the Archie client described in the previous section and a signficantly more sophisticated interface. The X11 interface makes it much easier for the average user to use this interface and significantly increases the user's productivity.

13.3 xrn

13.3.1 Introduction

xrn is an X11-based News reader. xrn assumes the existence of a NNTP 1.5 or newer News server on your network which handles all interaction with the Internet News system. xrn is merely a client which interacts with an NNTP server on your local network. There are several NNTP servers available. See uunet.uu.net for source code for these servers. This section will focus exclusively on the xrn client software system. Also see Chap. 6 for more information on News itself.

13.3.2 Usage

As stated earlier, xrn is an X11-based News reader system which runs on UNIX. xrn is a fairly simple system to use. Simply select the newsgroup you would like to use and proceed as you would with any other News reader.

You invoke xrn from the command line as follows:

```
xrn [options]
```

where options are many and varied. The man page describes all available options in great detail. It is not necessary, however, to use any of these options to interact quite nicely with the xrn system. Instead of documenting all options available with xrn, this section will merely outline those capabilities which make xrn very useful. Examine the xrn man page for more information on the large variety of options available with xrn.

xrn uses the file .newsrc to determine what newsgroups it needs to read. If this file does not exist, a new one will be created, and you will automatically join the news.announce.newusers group.

xrn has four modes of operation: add, newsgroup, all, and article. These modes control the basic operation of xrn itself. The add mode allows you to add new newsgroups to your reader. Once you have added a new group, it will remain and be automatically updated when you enter a new session of xrn. Newsgroup mode provides information on current newsgroups and gives you the ability to manipulate individual articles within newsgroups. The all mode provides you with a sorted list from which you can choose to add or manipulate newsgroups. Finally, the article mode allows you to manipulate groups or individual articles within a newsgroup. You can mark articles or groups of articles as read or unread as well as read individual articles. You can manually or automatically move from one mode to another based on your interaction with xrn.

This is a very basic introduction to xrn since it is extremely powerful and flexible. See the xrn man page for much more information on xrn usage and capabilities.

If you get an error message about not being able to connect to a server, you need to get the variable NNTPSERVER and reinvoke xrn. This must be a NetNews server that is accessible before invoking xrn.

13.3.3 Installation

The installation of xrn is relatively straightforward. Before you can invoke xrn, you must first ensure you have an NNTP server (version 1.5 or newer) on your network to which xrn can connect. Next, you need to

ensure that the NNTP server has been compiled with the XHDR option.

Once you have ensured that the NNTP server is configured and installed correctly or at least is accessible, you need to build xrn.

There is a makefile named Makefile which you can use as follows:

```
$ make clean
$ make USRLIBDIR=/usr/lpg/XII/lib/R5
$ make install
```

This will generate and install a clean copy of the xrn system into the /usr/local area since it is the default. If you have difficulties with this method, you may need to generate some new build files before proceeding with the compilation.

Once you have done this, you may install the software in /usr/local with the standard command:

```
$ make install
```

You can modify the behavior of xrn by changing the contents of the X11 resource file. See the Xresources.sample file for examples. See also the INSTALL.rs6k file for more details.

13.3.4 Conclusion

xrn provides a user-friendly interface to the Internet News system. While other readers provide a command line interface, xrn provides a true X11-based interface which makes it much easier to interact with the News system.

13.4 Xgopher

13.4.1 Introduction

Xgopher is an X11 interface to the Gopher system that was developed at the University of Minnesota. Gopher allows you to scan the Internet with a significantly enhanced interface from the normal FTP or telnet interface. The X11 interface to Gopher provides that much more useful information in an X11 format to make Gopher even more user friendly.

While Xgopher doesn't change the fundamental characteristics of the Gopher subsystem, it does provide a significantly more user-friendly interface to Gopher than the standard character interface. By providing all the advantages of the X11 interface, Xgopher will enhance your productivity and capabilities with Gopher.

This section does not discuss the function and capabilities of Gopher

but instead focuses on the Xgopher system. See Chap. 6 for more information on the Internet and Gopher itself.

13.4.2 Usage

The basic Xgopher screen is shown in Fig. 13.2. As you can see, Xgopher provides some basic functionality related to the Gopher subsystem. Buttons allow you to generate bookmarks, move around within the Gopher space, and perform other configuration-related commands. Through this X11 interface, you can quickly and easily begin to use the Gopher system on the Internet.

The syntax for Xgopher is:

```
xgopher [rootserver [server port]] [-toolkitoption...]
```

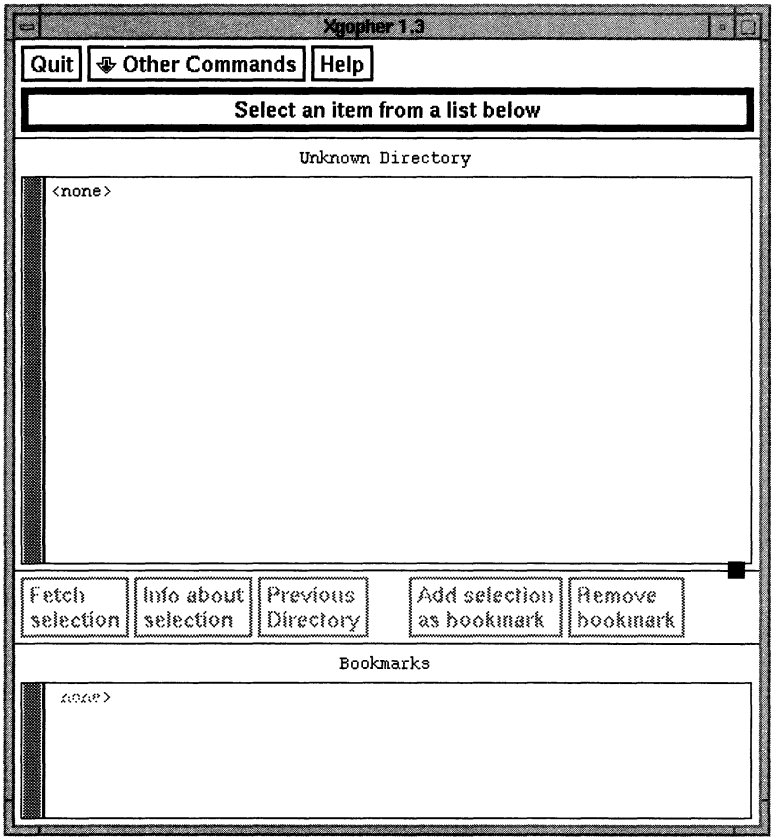

Figure 13.2 Xgopher screen.

where rootserver specifies the root Gopher server which to initially attach to.
server port specifies the RPC port to attach to for the Gopher service.
toolkitoption specifies an Xt tool kit option.

The rootserver and server port options are not required since you have configured a default Gopher server for initial connection to in the config.h file as documented below. However, if you wish to connect to a different server than the default, you can specify this on the command line. Finally, the toolkitoption options are the standard Xt X11 options. See the man page on Xt for more information on these options. Suffice it to say that they will merely affect the X11 characteristics of the Xgopher application.

While this section is not intended to provide an exhaustive discussion of Gopher, it is necessary to at least introduce the basic Gopher concepts. Gopher provides you with a standardized menu interface to resources on the Internet. Through the use of Gopher servers scattered throughout the world, you can access a wealth of information in a structured, organized way. By accessing a menu item, you transparently access a resource which may exist on a different machine or a different network halfway around the world. The power of Gopher is that you never have to know any of this to work effectively and to collect resources and information on your desktop. See Chap. 6 for more information on Gopher.

Xgopher works with a variety of other tools to enhance your Gopher capabilities. Tools such as xloadimage and tn3270 provide you with capabilities that the standard Gopher system does not have. This is what makes Xgopher so much more powerful than the standard Gopher client. All of the tools you need to use Xgopher effectively are contained either as a native part of AIX or are included on the accompanying CD.

Xgopher has a construct known as a bookmark. This is similar in many ways to an actual bookmark that has been used for years in paper books. By placing a bookmark at a particular location, you can quickly access that location again from within the same or a different session. This is a very powerful feature of Xgopher and is one that should be used extensively as you become more proficient with it.

Xgopher also provides you with the capability to search text indexes for specific information. By specifying a particular string, Xgopher will search all relevant indexes and present you with a list of documentation with that string in them. This is a powerful feature which can signicantly enhance your productivity with Xgopher.

As mentioned earlier, Xgopher provides you with a variety of interfaces to other non-vt100 types of terminals. It uses tn3270 to access

mainframe environments as well as tools such as xloadimage to view and manipulate images. You can also manipulate binary files and either execute or copy them to a local disk drive for later manipulation.

Finally, the options panel and X11 resources give you the ability to manipulate virtually all characteristics and behaviors of the Xgopher system. All the relevant resources are listed in the Xgoher.man file.

13.4.3 Installation

The installation of Xgopher is relatively straightforward. The commands are:

```
$ xmkmf
$ make depend
```

Once you have done this, you need to compare your newly generated Makefile with the file Makefile.aix32.r5 which was included in the distribution on the CD. You should note any changes relating to tool directories and other machine-specific information that may be different on your machine. Once you have made these changes, you may want to inspect the config.h file. This file contains information regarding the initial Gopher server to connect to as well as other information for Xgopher start-up. Modify these variables as you see fit before compiling the program. Once you have modified the appropriate variables within the config.h and Makefile, issue the command:

```
$ make
```

Next you may want to examine the application defaults file (an X11-based file most often known as the apps default file) Xgopher.ad. This contains definitions which the X11 server will use to initialize the Xgopher system on start-up. Review this file and modify any variables which you may want changed. There is certainly nothing that you should have to change here, and you can ignore this step if you would like.

Next, you need to make this apps default file known to your X11 server with the command:

```
$ xrdb -merge Xgopher.ad
```

If you have trouble with this, you can set the environment variable XENVIRONMENT to point to this with commands like:

```
$ XENVIRONMENT=$XENVIRONMENT:`pwd`/Xgopher.ad
```

or

```
export XENVIRONMENT
```

Either of these will make the X11 server aware of the existance of this apps default file. Finally, you can install the Xgopher application with the command:

```
$ make install
```

If you want to modify the installation directories you can either include the specific directory on the make command line or modify the INSTALL directory in the Makefile.

This is all there is to creating Xgopher.

13.4.4 Conclusion

Xgopher provides an X11-based interface to the Gopher system on the Internet. By providing all of the capabilities of Gopher in an X11 interface, Xgopher signficantly enhances your productivity in a Gopher environment.

Through the use of a menu-driven interface to the Internet and enhanced tools which provide image manipulation and enhanced file handling, Xgopher is a must-use tool for any sophisticated developer.

13.5 Mosaic

13.5.1 Introduction

Mosaic is a multimedia viewer to the Internet. Through the use of a new network paradigm known as the World Wide Web (WWW), Mosaic provides a new type of view and new, more powerful capabilities to get to information on the Internet. Mosaic was written at the National Center for Supercomputer Applications (better known as NCSA) at the University of Illinois at Champaign and is freely available on the Internet from a variety of locations. Due to licensing restrictions, it is not included on the CD accompanying this book. It is freely available, however, to anyone who wishes to use it for personal use. So, the first thing for you to do is to get a freely available copy from the Internet. Once you have done this, you should proceed with the rest of this chapter.

This section will not discuss details of Mosaic but will instead present a brief description of its capabilities and relevant information for AIX. You will need to reference other documentation and books for specific information on Mosaic.

13.5.2 Usage

There is really very little to invoking Mosaic from the command line. Simply type:

```
$ xmosaic
```

There are a variety of Xt options related to the X11 tool kit. See the man pages for Xt for more information. All other information, including start-up information, can be controlled and modified from within Mosaic itself.

The initial Mosaic screen is shown in Fig. 13.3. As you can see, this screen presents a lot of information, but there are a few portions which deserve special mention. The first is the Document Title bar. This pre-

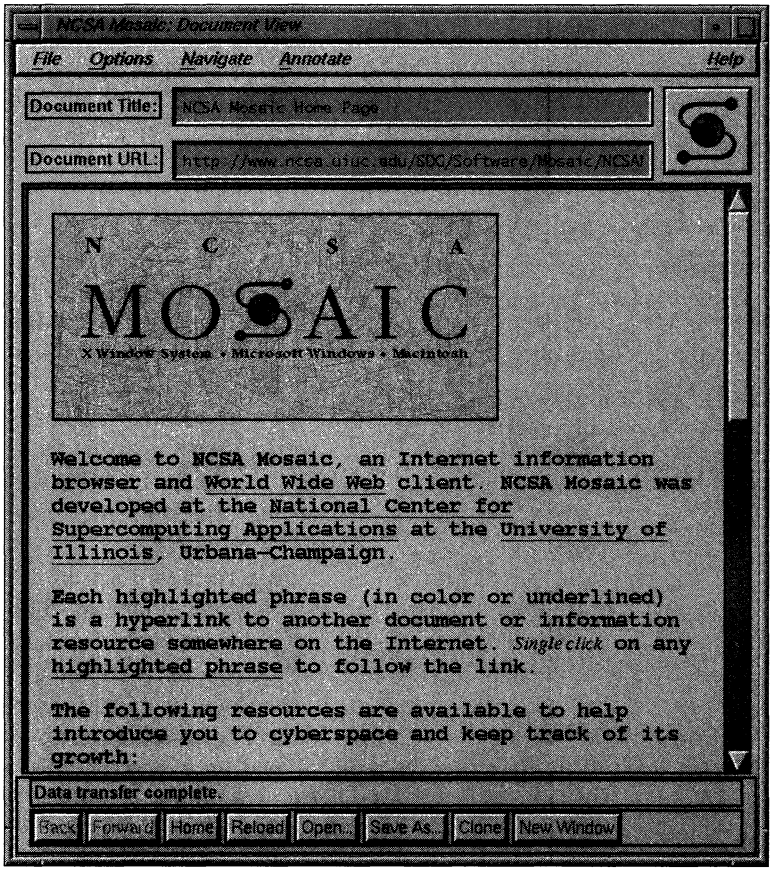

Figure 13.3 The Mosaic screen.

sents the title of the current document you are viewing. Mosaic is, in actuality, only a front end to the WWW and uses other tools such as image viewers to actually manipulate the data coming onto the local display. Most of these tools are provided on the accompanying CD; however, there may be other tools which you want to use depending on your specific use of Mosaic. See the Mosaic documentation for more details.

The next line represents the Document URL. URL stands for Universal Resource Locator and is a specification which tells Mosaic where to get a specific resource and how. The basic syntax of this Document URL is:

```
http://www.ncsa.uiuc.edu/SDC/Software/Mosaic/NCSA.html
```

The http specifies the hypertext transfer protocol; it is the standard used by all Mosaic implementations to transfer information and coordinate all conversations between server and client. You can specify other protocols such as FTP to access other types of services if you so desire.

The next string, www.ncsa.uiuc.edu, specifies the Mosaic server to which the Mosaic client is connected. This is a machine name on the Internet which is serving the WWW. By changing this specification, you can move from one machine to another transparently.

Finally, SDC/Software/Mosaic/NCSA.html specifies a file on www.ncsa.uiuc.edu which contains all the information you are viewing on your screen. It is important to understand that Mosaic is simply a viewer to information on the Internet and has very little intelligence by itself. The files which Mosaic uses are formatted in a special language known as HyperText Markup Language (html). This is a specific language which has many similarities to Standardized General Markup Language (SGML), which is becoming the de facto standard for document and image formatting throughout the world. html is well documented and actually quite easy to use. You can download the html file for anything you are viewing by using the Options pulldown menu on the Mosaic client interface.

The rest of the main screen is the main article which you are viewing. This can consist of images and text intermixed in any format. The important thing to note about the text is that if the text is underlined or highlighted, it is a hypertext link to a different resource on the Internet. By simply clicking on this piece of underlined or highlighted text, you will immediately transfer to that resource, and the appropriate html document will be downloaded and viewed automatically. Remember that this document may be on a different machine or network anywhere around the world.

Once you have done this, you will see some of the other buttons at the bottom of the screen darken. This means that they are now avail-

able for your use. You can move back a document by simply clicking back, can move forward by clicking on the forward button, and so forth. You can also go to your original page known as the home page. This will allow you to go back where you started with the click of a mouse.

There are many other interesting aspects of Mosaic which are beyond the scope of this book; however, it is hoped that this has whetted your appetite for more information on Mosaic and its capabilities. See any number of good books on the subject as well as the documentation accompanying the Mosaic distribution.

It should be noted that there are a variety of commercial ports of the Mosaic system available today. Each has different strengths and weaknesses relative to the freeware version; however, the NCSA version is, overall, as good as the others and should be your first attempt at utilizing the WWW with Mosaic.

13.5.3 Installation

The installation of Mosaic is very straightforward:

```
$ cp Makefile.RS6k Makefile
$ make
$ make install
```

If you experience any problems with missing X11 libraries and include files, you may have to issue the xmkmf command before building. To do this, use the command:

```
$ xmkmf
```

This generates a new Makefile. Next you need to compare the newly created Makefile with Makefile.RS6k. Make any necessary changes to ensure that the AIX-specific information for Mosaic is contained in Makefile. Next issue the commands:

```
$ make
$ make install
```

This will generate the Mosaic system for you.

13.5.4 Conclusion

Mosaic is clearly the fastest growing tool in use of the Internet. By providing multimedia/hypermedia capabilities to the Internet, Mosaic has opened up an entirely new paradigm for information access and exchange on the Internet. No developer will be able to afford to be without it.

Appendix

A

How to Get Software from the CD

This CD was designed and tested for AIX 3.2.*x* and AIX 4.1. Depending on the driver software with your operating system, the filenames and extensions may be uppercase, lowercase, or mixed case. Unfortunately, because AIX 3.2.*x* doesn't support Rock Ridge extensions, I was forced to use ISO 9660 compliant filenames for all compressed tar files and thus lose some of the power and capabilities of executing this software directly from the CD on AIX 3.2.*x*. This was most definitely a compromise. In addition, for all source code and executables on the disk, there are compressed tar files for all software under ./EXEC and ./SRC. Copy these if you have trouble accessing the native software.

To mount the CD on an AIX machine, use the command:

```
# mount -r -v cdrfs /dev/cd0 /cdrom
```

Note that if you have more than one CD device on your machine, you may have to use a different device than /dev/cd0. Also, /cdrom can be replaced with any empty directory. You should coordinate with your system administrator on these issues if you have any questions.

If you are running AIX 3.2.*x*, you will need to copy the appropriate file from the CD onto a hard disk and rename it. All files follow the naming convention outlined below except that they are in lowercase.

A filename without an extension is simply that filename and can be copied directly as named to a hard disk. A filename with an extension of TAZ is a compressed tar file. When you copy the file to the hard disk, replace the TAZ extension with tar.Z. For example, the file for bash is BASH1142.TAZ. Copy this to the hard disk with a command like:

```
$ cp /cdrom/EXEC/BASH1142.TAZ /usr/local/bash1142.tar.Z
```

From here you need to uncompress and untar the file with the commands:

```
$ uncompress bash1142.tar
$ tar xvf bash1142.tar
```

This will uncompress and unwind the tar file in your current directory.

You can examine the contents of the tar file with the command:

```
$ tar tvf bash1142.tar
```

There are two areas of special interest on the CD. The first is the binary area under the EXEC directory; the other is the product source code directories under the SRC directory.

If you examine the ./EXEC/USR/LOCAL/BIN, you will find all the precompiled files for all the included software packages. These filenames have all been included *unmodified* for your ease of use, and you can copy them directly from the CD to a hard disk and run them. Under AIX 3.2.*x* the driver software for the CD does not support the standard UNIX filename convention. This means that filenames may be truncated or may contain unusual characters. You can either access these files as presented or access them through the compressed tar files on the CD as documented above. In fact, if you desire, you can execute them directly from the CD. With AIX 4.1, you should be able to view these. These filenames may cause problems on AIX 3.2.*x* because they do not necessarily conform to the ISO 9660 standard. Note that these are also included in the compressed tar files in the EXEC directory, so you can get at them this way as well.

In the ./SRC directory, you will see both files with the .TAZ suffix, which designates compressed tar files, and you will see files which will vary from machine to machine depending on your driver. You can treat the .TAZ files the same as those described above. The other files are directories and you should be able to move into these directories and examine the files directly. You may see strange filenames since these also do not conform to the ISO 9660 specification. This was done to allow machines which support Rock Ridge extensions (such as AIX 4.1) to manipulate these files just as if they were a UNIX filesystem. On certain OSs, including AIX 3.2.*x,* you may either copy these files directly to the hard disk or move the equivalent compressed tar files from the ./src directory as described above.

The only file that does not conform to the conventions listed above is ./SRC/TCL74B4.PAZ. The expanded filename of this file is tcl74b4.patch.Z. This is a compressed patch file for the patch utility.

Appendix

B

General Notes About the Software on the CD

This appendix outlines some general notes and information concerning the software on the accompanying CD. This software was pulled from the Internet in a variety of ways and from a variety of locations.

The general installation sections in this book assume that you are using the software provided on the CD. The general differences between the software on the CD and the ones you can pull from the Internet are documented below. Keep these in mind as you go to the Internet for more software and systems. You may have to modify the build behavior depending on where you get the software.

B.1 Archives

Some of the information on this CD was retrieved from the machine aixpdslib.seas.ucla.edu. This is a very large AIX software server maintained at UCLA. It contains a variety of software systems built for the AIX environment. You should use this machine to retrieve systems for your AIX machine if at all possible. They have done work building special makefiles (see the next section) and documenting strange and unusual behavior when building or running a particular application on AIX.

There are other archives, particularly prep.ai.mit.edu, which contain a large number of software systems. You should feel free to pull from these locations to get these types of software systems; however, they won't typically have the types of AIX-specific information that aixpdslib.seas.ucla.edu has; therefore, you may have more work to build a particular product.

As you know, there are thousands of UNIX server nodes on the Internet from which you can pull software systems which may be useful to you. The only way for you to discover more is by investigation. Good luck and have fun.

B.2 Makefiles and Installation Notes

Some of the systems will have a file Makefile.rs6k or Makefile.RS6k, while others will not. This turns out to be the first thing you should look for. If you find the Makefile.RS6k files, you should then look for a file INSTALL.rs6k or INSTALL.RS6k, which will document many issues with respect to the particular AIX makefile in question. These are very powerful tools which have been modified and provided by the people who maintain the aixpdslib.seas.ucla.edu system.

Note, however, that the AIX-specific makefiles may or may not work. There are a variety of system-specific things that are built into them, including X11 locations (see below), tool locations, choices (see below), and other oddities. If you have trouble building the software systems with the provided makefiles, you may need to use the xmkmf facility to generate new, machine-specific makefiles for your particular machine configuration. See the INSTALL files for more details on this.

B.3 xmkmf/imake

xmkmf and imake are documented in Chap. 7 of this book; however, there are specific issues related to the software on this CD which deserve mention in this appendix. Some of the software systems on the accompanying CD contain imakefiles which can produce new, machine-specific makefiles. Several of the INSTALL files suggest that you may want to use xmkmf to generate new makefiles before building the software system.

If you have trouble with unresolved symbols or files that cannot be found, use the xmkmf facility provided with this CD to generate new, machine-specific makefiles from which you can then generate your software. This may save you a significant amount of manual effort in modifying the makefiles and troubleshooting the results.

B.4 M.I.T. and IBM X11 Libraries and Include Files

Many of the tools included on this CD use the M.I.T. X11 libraries as the default against which to link. The default location for these libraries is /usr/local/XIIR5. Because of this, many of the Makfile.RS6k files

reference dependencies in this directory. There are two ways you can handle this. A portion of the X11R5 libraries in the /usr/local area has been included on the CD. You can also change the references from /usr/local/X11R5 to /usr/lpp/X11. This will point the makefiles to the location on the RS/6000 where the X11R5 libraries exist. Note that you need to be running AIXWindows 1.2 before you can ensure that you are running X11R5.

Most tools assume you are running X11R5. If you are running an earlier or later release, you will need to see the documentation for details on exactly what you need to do to make these systems build correctly.

B.5 Tool Locations and Choices

Many of the build scripts assume that you are going to use GNU tools such as gcc, bison, and gawk to build your applications. This may or may not cause trouble in your build procedures. Check the beginning definitions in the makefiles before building for the first time to ensure that you have all referenced tools installed. You can use many of the tools provided with AIX, such as cc, sed, and awk; however, you may or may not have strange problems depending on the dependence of the makefile logic on the functionality of the tools it is using. Watch out for this one.

B.6 Common Errors

People have been having a problem with dynamic links when running the prebuilt version of X-stuffs from this library. The typical error messages are:

Could not load program [program_name].

Member shr4.o not found or file not an archive.

Member shr4.o not found or file not an archive.

Could not load library libXt.a[shr4.o].

Error was: No such file or directory.

They occur because the programs were built using X11 M.I.T. libraries which are not compatible with those of IBM. If this is the case, get the compressed tar file of the source code and recompile on your system using your libraries. This is explained on the aixpdslib.seas.ucla.edu system and documents a typical problem encountered on systems retrieved from their server. It is related to the X11 discussion above. Watch out for this one.

There is also a relatively common error relating to the XtStrings reference. You may see this as an unresolved reference in a variety of systems when you go to link the final executable. If this occurs, you may need to modify either the imakefile or the appropriate makefile to include the macro definition XTSTRINGDEFINES. If you include this with the compilation directives, this will usually remove this linking problem. Try this or see the INSTALL or PROBLEMS files for more details.

Appendix

C

General Licenses

C.1 GNU General Public License

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING,
DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.> Copyright (C) 19yy <name of author>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) 19yy name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'. This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program 'Gnomovision' (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989 Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

C.2 GNU Library General Public License

GNU LIBRARY GENERAL PUBLIC LICENSE
Version 2, June 1991

Copyright (C) 1991 Free Software Foundation, Inc. 675 Mass Ave, Cambridge, MA 02139, USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

[This is the first released version of the library GPL. It is numbered 2 because it goes with version 2 of the ordinary GPL.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users.

This license, the Library General Public License, applies to some specially designated Free Software Foundation software, and to any other libraries whose authors decide to use it. You can use it for your libraries, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library, or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link a program with the library, you must provide complete object files to the recipients so that they can relink them with the library, after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

Our method of protecting your rights has two steps: (1) copyright the library, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the library.

Also, for each distributor's protection, we want to make certain that everyone understands that there is no warranty for this free library. If the library is modified by someone else and passed on, we want its recipients to know that what they have is not the original version, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that companies distributing free software will individually obtain patent licenses, thus in effect transforming the program into proprietary software. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License, which was designed for utility programs. This license, the GNU Library General Public License, applies to certain designated libraries. This license is quite different from the ordinary one; be sure to read it in full, and don't assume that anything in it is the same as in the ordinary license.

The reason we have a separate public license for some libraries is that they blur the distinction we usually make between modifying or adding to a program and simply using it. Linking a program with a library, without changing the library, is in some sense simply using the library, and is analogous to running a utility program or application program. However, in a textual and legal sense, the linked executable is a combined work, a derivative of the original library, and the ordinary General Public License treats it as such.

Because of this blurred distinction, using the ordinary General Public License for libraries did not effectively promote software sharing, because most developers did not use the libraries. We concluded that weaker conditions might promote sharing better.

However, unrestricted linking of non-free programs would deprive the users of those programs of all benefit from the free status of the libraries themselves. This Library General Public License is intended to permit developers of non-free programs to use free libraries, while preserving your freedom as a user of such programs to change the free libraries that are incorporated in them. (We have not seen how to achieve this as regards changes in header files, but we have achieved it as regards changes in the actual functions of the Library.) The hope is that this will lead to faster development of free libraries.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, while the latter only works together with the library.

Note that it is possible for a library to be covered by the ordinary General Public License rather than by this special one.

GNU LIBRARY GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING,
DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Library General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) The modified work must itself be a software library.

b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered inde-

pendent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also compile or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit

modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

c) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

d) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Library General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT

LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix: How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the library's name and a brief idea of what it does.> Copyright (C) <year> <name of author>

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Library General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Library General Public License for more details.

You should have received a copy of the GNU Library General Public License along with this library; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the library 'Frob' (a library for tweaking knobs) written by James Random Hacker.

<signature of Ty Coon>, 1 April 1990 Ty Coon, President of Vice

That's all there is to it!

C.3 Most Recent GETTING.GNU.SOFTWARE File

-*- text -*- Getting GNU Software, 14 May 94

* GNU and the Free Software Foundation

Project GNU is organized as part of the Free Software Foundation, Inc. The Free Software Foundation has the following goals: 1) to create GNU as a full development/operating system. 2) to distribute GNU and other useful software with source code and permission to copy and redistribute.

Further information on the rationale for GNU is in file '/pub/gnu/GNUinfo/GNU' (all files referred to are on the Internet host prep.ai.mit.edu).

Information on GNU Internet mailing lists and gnUSENET newsgroups can be found in '/pub/gnu/GNUinfo/MAILINGLISTS',

* How To Get The Software

The easiest way to get a copy of the distribution is from someone else who has it. You need not ask for permission to do so, or tell any one else; just copy it. The second easiest is to ftp it over the Internet. The third easiest way is to uucp it. Ftp and uucp information is in '/pub/gnu/GNUinfo/FTP'.

If you cannot get a copy any of these ways, or if you would feel more confident getting copies straight from us, or if you would like to get some funds to us to help in our efforts, you can order one from the Free Software Foundation. See '/pub/gnu/GNUinfo/DISTRIB' and '/pub/gnu/GNUinfo/ORDERS'.

* What format are the *.gz files in?

Because the unix 'compress' utility is patented (by two separate patents, in fact), we cannot use it; it's not free software.

Therefore, the GNU Project has chosen a new compression utility, 'gzip', which is free of any known software patents and which tends to compress better anyway. As of March 1993, all compressed files in the GNU anonymous FTP area, 'prep.ai.mit.edu:/pub/gnu', have been converted to the new format. Files compressed with this new compression program end in '.gz' (as opposed to 'compress'-compressed files, which end in '.Z').

Gzip can uncompress 'compress'-compressed files and 'pack'-compressed files (which end in '.z'). This is possible because the various decompression algorithms are not patented—only compression is.

The gzip program is available from any GNU mirror site (see '/pub/gnu/GNUinfo/FTP' for a list of mirror sites) in shar, tar, or gzipped tar format (for those who already have a prior version of gzip and want faster data transmission). It works on virtually every unix system, MSDOS, OS/2, and VMS.

* Available Software

** GNU Emacs

The GNU Emacs distribution includes:

—manual source in TeX format.

—an enhanced regex (regular expression) library.

See files '/pub/gnu/GNUinfo/MACHINES*' for the status of porting Emacs to various machines and operating systems.

** C Scheme - a block structured dialect of LISP.

The Free Software Foundation distributes C Scheme for the MIT Scheme Project on its Scheme tapes. The full ftp distribution can be gotten via anonymous FTP from altdorf.ai.mit.edu in directory /archive.

Problems with the C Scheme distribution and its ftp distribution should be referred to: g-cscheme@martigny.ai.mit.edu. There are two general mailing lists: heme@martigny.ai.mit.eduand heme@mc.lcs.mit.edu. Send requests to join either list to: heme-request@martigny.ai.mit.edu or heme-request@mc.lcs.mit.edu.

** Other GNU Software

A full list of available software are in '/pub/gnu/GNUinfo/ORDERS' and '/pub/gnu/DESCRIPTIONS'.

* No Warranties

We distribute software in the hope that it will be useful, but without any warranty. No author or distributor of this software accepts responsibility to anyone for the consequences of using it or for whether it serves any particular purpose or works at all, unless he says so in writing.

* If You Like The Software

If you like the software developed and distributed by the Free Software Foundation, please express your satisfaction with a donation. Your donations will help to support the foundation and make our future efforts successful, including a complete development and operating system, called GNU (Gnu's Not Un*x), which will run Un*x user programs. Please note that donations and funds raised by selling tapes, CD-ROMs, and floppy diskettes are the major source of funding for our work.

For more information on GNU and the Foundation, contact us at Internet address gnu@prep.ai.mit.edu or the foundation's US Mail address found in file '/pub/gnu/GNUinfo/ORDERS'.

C.4 Author's Disclaimer

The author and distributor of this software is in no way responsible for anything related to either the performance or distribution of the software on the accompanying CD. The author is also not responsible for any maintenance or support activities on anything related to the software or the distribution of the software. The author also stands by all Copyright and permission notices of all software on the CD.

C.5 M.I.T.'s Disclaimer

Copyright 1991 by the Massachusetts Institute of Technology.

Permission to use, copy, modify, and distribute this document for any purpose and without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies, and that the name of MIT not be used in advertising or publicity pertaining to this document without specific, written prior permission. MIT makes no representations about the suitability of this document for any purpose. It is provided "as is" without express or implied warranty.

C.6 The Regents of the University of California Disclaimer

Flex carries the copyright used for BSD software, slightly modified because it originated at the Lawrence Berkeley (not Livermore!) Laboratory, which operates under a contract with the Department of Energy:

Copyright (c) 1990 The Regents of the University of California. All rights reserved.

This code is derived from software contributed to Berkeley by Vern Paxson.

The United States Government has rights in this work pursuant to contract no. DE-AC03-76SF00098 between the United States Department of Energy and the University of California.

Redistribution and use in source and binary forms are permitted provided that: (1)

source distributions retain this entire copyright notice and comment, and (2) distributions including binaries display the following acknowledgement: "This product includes software developed by the University of California, Berkeley and its contributors" in the documentation or other materials provided with the distribution and in all advertising materials mentioning features or use of this software. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

This basically says "do whatever you please with this software except remove this notice or take advantage of the University's (or the flex authors') name."

Note that the "flex.skel" scanner skeleton carries no copyright notice. You are free to do whatever you please with scanners generated using flex; for them, you are not even bound by the above copyright.

C.7 AT&T/BellCore Copyright

```
**************************************************************************
```

Copyright 1990 by AT&T Bell Laboratories and Bellcore.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that the copyright notice and this permission notice and warranty disclaimer appear in supporting documentation, and that the names of AT&T Bell Laboratories or Bellcore or any of their entities not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

AT&T and Bellcore disclaim all warranties with regard to this software, including all implied warranties of merchantability and fitness. In no event shall AT&T or Bellcore be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with the use or performance of this software.

```
**************************************************************************/
```

C.8 pbmplus Copyright

COPYRIGHTS

All the software in this package, whether by me or by a contributor, has a copyright similar to this one:

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. This software is provided "as is" without express or implied warranty.

Many people get confused by this legalese, especially the part about "without fee." Does this mean you can't charge for any product that uses PBMPLUS? No. All it means is that you don't have to pay me. You can do what you want with this software. Build it into your package, steal code from it, whatever. Just be sure to let people know where it came from.

C.9 gnuplot Copyright

```
/*
* Copyright (C) 1986 - 1993 Thomas Williams, Colin Kelley
*
* Permission to use, copy, and distribute this software and its
* documentation for any purpose with or without fee is hereby granted,
* provided that the above copyright notice appear in all copies and
* that both that copyright notice and this permission notice appear
* in supporting documentation.
*
* Permission to modify the software is granted, but not the right to
* distribute the modified code. Modifications are to be distributed
* as patches to released version.
*
* This software is provided "as is" without express or implied warranty.
*
*
* AUTHORS
*
* Original Software:
* Thomas Williams, Colin Kelley.
*
* Gnuplot 2.0 additions:
* Russell Lang, Dave Kotz, John Campbell.
*
* Gnuplot 3.0 additions:
* Gershon Elber and many others.
*
* Send your comments or suggestions to
* info-gnuplot@dartmouth.edu.
* This is a mailing list; to join it send a note to
* info-gnuplot-request@dartmouth.edu.
* Send bug reports to
* bug-gnuplot@dartmouth.edu.
*/
```

C.10 The GNU Manifesto

The GNU Manifesto

Copyright (C) 1985 Richard M. Stallman (Copying permission notice at the end.)

What's GNU? Gnu's Not Unix!

GNU, which stands for Gnu's Not Unix, is the name for the complete Unix-compatible software system which I am writing so that I can give it away free to everyone who can use it. Several other volunteers are helping me. Contributions of time, money, programs and equipment are greatly needed.

So far we have an Emacs text editor with Lisp for writing editor commands, a source level debugger, a yacc-compatible parser generator, a linker, and around 35 utilities. A shell (command interpreter) is nearly completed. A new portable optimizing C compiler has compiled itself and may be released this year. An initial kernel exists but many more features are needed to emulate Unix. When the kernel and compiler are finished, it will be possible to distribute a GNU system suitable for program development. We will use @TeX{} as our text formatter, but an nroff is being worked on. We will use the free, port-

able X window system as well. After this we will add a portable Common Lisp, an Empire game, a spreadsheet, and hundreds of other things, plus on-line documentation. We hope to supply, eventually, everything useful that normally comes with a Unix system, and more.

GNU will be able to run Unix programs, but will not be identical to Unix. We will make all improvements that are convenient, based on our experience with other operating systems. In particular, we plan to have longer filenames, file version numbers, a crashproof file system, filename completion perhaps, terminal-independent display support, and perhaps eventually a Lisp-based window system through which several Lisp programs and ordinary Unix programs can share a screen. Both C and Lisp will be available as system programming languages. We will try to support UUCP, MIT Chaosnet, and Internet protocols for communication.

GNU is aimed initially at machines in the 68000/16000 class with virtual memory, because they are the easiest machines to make it run on. The extra effort to make it run on smaller machines will be left to someone who wants to use it on them.

To avoid horrible confusion, please pronounce the 'G' in the word 'GNU' when it is the name of this project.

Who Am I?

I am Richard Stallman, inventor of the original much-imitated EMACS editor, formerly at the Artificial Intelligence Lab at MIT. I have worked extensively on compilers, editors, debuggers, command interpreters, the Incompatible Timesharing System and the Lisp Machine operating system. I pioneered terminal-independent display support in ITS. Since then I have implemented one crashproof file system and two window systems for Lisp machines, and designed a third window system now being implemented; this one will be ported to many systems including use in GNU. [Historical note: The window system project was not completed; GNU now plans to use the X window system.]

Why I Must Write GNU

I consider that the golden rule requires that if I like a program I must share it with other people who like it. Software sellers want to divide the users and conquer them, making each user agree not to share with others. I refuse to break solidarity with other users in this way. I cannot in good conscience sign a nondisclosure agreement or a software license agreement. For years I worked within the Artificial Intelligence Lab to resist such tendencies and other inhospitalities, but eventually they had gone too far: I could not remain in an institution where such things are done for me against my will.

So that I can continue to use computers without dishonor, I have decided to put together a sufficient body of free software so that I will be able to get along without any software that is not free. I have resigned from the AI lab to deny MIT any legal excuse to prevent me from giving GNU away.

Why GNU Will Be Compatible with Unix

Unix is not my ideal system, but it is not too bad. The essential features of Unix seem to be good ones, and I think I can fill in what Unix lacks without spoiling them. And a system compatible with Unix would be convenient for many other people to adopt.

How GNU Will Be Available

GNU is not in the public domain. Everyone will be permitted to modify and redistribute GNU, but no distributor will be allowed to restrict its further redistribution. That is to say, proprietary modifications will not be allowed. I want to make sure that all versions of GNU remain free.

Why Many Other Programmers Want to Help

I have found many other programmers who are excited about GNU and want to help.

Many programmers are unhappy about the commercialization of system software. It

may enable them to make more money, but it requires them to feel in conflict with other programmers in general rather than feel as comrades. The fundamental act of friendship among programmers is the sharing of programs; marketing arrangements now typically used essentially forbid programmers to treat others as friends. The purchaser of software must choose between friendship and obeying the law. Naturally, many decide that friendship is more important. But those who believe in law often do not feel at ease with either choice. They become cynical and think that programming is just a way of making money.

By working on and using GNU rather than proprietary programs, we can be hospitable to everyone and obey the law. In addition, GNU serves as an example to inspire and a banner to rally others to join us in sharing. This can give us a feeling of harmony which is impossible if we use software that is not free. For about half the programmers I talk to, this is an important happiness that money cannot replace.

How You Can Contribute

I am asking computer manufacturers for donations of machines and money. I'm asking individuals for donations of programs and work.

One consequence you can expect if you donate machines is that GNU will run on them at an early date. The machines should be complete, ready to use systems, approved for use in a residential area, and not in need of sophisticated cooling or power.

I have found very many programmers eager to contribute part-time work for GNU. For most projects, such part-time distributed work would be very hard to coordinate; the independently-written parts would not work together. But for the particular task of replacing Unix, this problem is absent. A complete Unix system contains hundreds of utility programs, each of which is documented separately. Most interface specifications are fixed by Unix compatibility. If each contributor can write a compatible replacement for a single Unix utility, and make it work properly in place of the original on a Unix system, then these utilities will work right when put together. Even allowing for Murphy to create a few unexpected problems, assembling these components will be a feasible task. (The kernel will require closer communication and will be worked on by a small, tight group.)

If I get donations of money, I may be able to hire a few people full or part time. The salary won't be high by programmers' standards, but I'm looking for people for whom building community spirit is as important as making money. I view this as a way of enabling dedicated people to devote their full energies to working on GNU by sparing them the need to make a living in another way.

Why All Computer Users Will Benefit

Once GNU is written, everyone will be able to obtain good system software free, just like air.

This means much more than just saving everyone the price of a Unix license. It means that much wasteful duplication of system programming effort will be avoided. This effort can go instead into advancing the state of the art.

Complete system sources will be available to everyone. As a result, a user who needs changes in the system will always be free to make them himself, or hire any available programmer or company to make them for him. Users will no longer be at the mercy of one programmer or company which owns the sources and is in sole position to make changes.

Schools will be able to provide a much more educational environment by encouraging all students to study and improve the system code. Harvard's computer lab used to have the policy that no program could be installed on the system if its sources were not on public display, and upheld it by actually refusing to install certain programs. I was very much inspired by this.

Finally, the overhead of considering who owns the system software and what one is or is not entitled to do with it will be lifted.

Arrangements to make people pay for using a program, including licensing of copies,

always incur a tremendous cost to society through the cumbersome mechanisms necessary to figure out how much (that is, which programs) a person must pay for. And only a police state can force everyone to obey them. Consider a space station where air must be manufactured at great cost: charging each breather per liter of air may be fair, but wearing the metered gas mask all day and all night is intolerable even if everyone can afford to pay the air bill. And the TV cameras everywhere to see if you ever take the mask off are outrageous. It's better to support the air plant with a head tax and chuck the masks.

Copying all or parts of a program is as natural to a programmer as breathing, and as productive. It ought to be as free.

Some Easily Rebutted Objections to GNU's Goals

"Nobody will use it if it is free, because that means they can't rely on any support." "You have to charge for the program to pay for providing the support."

If people would rather pay for GNU plus service than get GNU free without service, a company to provide just service to people who have obtained GNU free ought to be profitable.

We must distinguish between support in the form of real programming work and mere handholding. The former is something one cannot rely on from a software vendor. If your problem is not shared by enough people, the vendor will tell you to get lost.

If your business needs to be able to rely on support, the only way is to have all the necessary sources and tools. Then you can hire any available person to fix your problem; you are not at the mercy of any individual. With Unix, the price of sources puts this out of consideration for most businesses. With GNU this will be easy. It is still possible for there to be no available competent person, but this problem cannot be blamed on distribution arrangements. GNU does not eliminate all the world's problems, only some of them. Meanwhile, the users who know nothing about computers need handholding: doing things for them which they could easily do themselves but don't know how.

Such services could be provided by companies that sell just hand-holding and repair service. If it is true that users would rather spend money and get a product with service, they will also be willing to buy the service having got the product free. The service companies will compete in quality and price; users will not be tied to any particular one. Meanwhile, those of us who don't need the service should be able to use the program without paying for the service.

"You cannot reach many people without advertising, and you must charge for the program to support that." "It's no use advertising a program people can get free."

There are various forms of free or very cheap publicity that can be used to inform numbers of computer users about something like GNU. But it may be true that one can reach more microcomputer users with advertising. If this is really so, a business which advertises the service of copying and mailing GNU for a fee ought to be successful enough to pay for its advertising and more. This way, only the users who benefit from the advertising pay for it. On the other hand, if many people get GNU from their friends, and such companies don't succeed, this will show that advertising was not really necessary to spread GNU. Why is it that free market advocates don't want to let the free market decide this?

"My company needs a proprietary operating system to get a competitive edge."

GNU will remove operating system software from the realm of competition. You will not be able to get an edge in this area, but neither will your competitors be able to get an edge over you. You and they will compete in other areas, while benefiting mutually in this one. If your business is selling an operating system, you will not like GNU, but that's tough on you. If your business is something else, GNU can save you from being pushed into the expensive business of selling operating systems.

I would like to see GNU development supported by gifts from many manufacturers and users, reducing the cost to each.

"Don't programmers deserve a reward for their creativity?"

If anything deserves a reward, it is social contribution. Creativity can be a social con-

tribution, but only in so far as society is free to use the results. If programmers deserve to be rewarded for creating innovative programs, by the same token they deserve to be punished if they restrict the use of these programs.

"Shouldn't a programmer be able to ask for a reward for his creativity?"

There is nothing wrong with wanting pay for work, or seeking to maximize one's income, as long as one does not use means that are destructive. But the means customary in the field of software today are based on destruction.

Extracting money from users of a program by restricting their use of it is destructive because the restrictions reduce the amount and the ways that the program can be used. This reduces the amount of wealth that humanity derives from the program. When there is a deliberate choice to restrict, the harmful consequences are deliberate destruction.

The reason a good citizen does not use such destructive means to become wealthier is that, if everyone did so, we would all become poorer from the mutual destructiveness. This is Kantian ethics; or, the Golden Rule. Since I do not like the consequences that result if everyone hoards information, I am required to consider it wrong for one to do so. Specifically, the desire to be rewarded for one's creativity does not justify depriving the world in general of all or part of that creativity.

"Won't programmers starve?"

I could answer that nobody is forced to be a programmer. Most of us cannot manage to get any money for standing on the street and making faces. But we are not, as a result, condemned to spend our lives standing on the street making faces, and starving. We do something else.

But that is the wrong answer because it accepts the questioner's implicit assumption: that without ownership of software, programmers cannot possibly be paid a cent. Supposedly it is all or nothing.

The real reason programmers will not starve is that it will still be possible for them to get paid for programming; just not paid as much as now.

Restricting copying is not the only basis for business in software. It is the most common basis because it brings in the most money. If it were prohibited, or rejected by the customer, software business would move to other bases of organization which are now used less often. There are always numerous ways to organize any kind of business.

Probably programming will not be as lucrative on the new basis as it is now. But that is not an argument against the change. It is not considered an injustice that sales clerks make the salaries that they now do. If programmers made the same, that would not be an injustice either. (In practice they would still make considerably more than that.)

"Don't people have a right to control how their creativity is used?"

"Control over the use of one's ideas" really constitutes control over other people's lives; and it is usually used to make their lives more difficult.

People who have studied the issue of intellectual property rights carefully (such as lawyers) say that there is no intrinsic right to intellectual property. The kinds of supposed intellectual property rights that the government recognizes were created by specific acts of legislation for specific purposes.

For example, the patent system was established to encourage inventors to disclose the details of their inventions. Its purpose was to help society rather than to help inventors. At the time, the life span of 17 years for a patent was short compared with the rate of advance of the state of the art. Since patents are an issue only among manufacturers, for whom the cost and effort of a license agreement are small compared with setting up production, the patents often do not do much harm. They do not obstruct most individuals who use patented products.

The idea of copyright did not exist in ancient times, when authors frequently copied other authors at length in works of non-fiction. This practice was useful, and is the only way many authors' works have survived even in part. The copyright system was created expressly for the purpose of encouraging authorship. In the domain for which it was invented—books, which could be copied economically only on a printing press—it did little harm, and did not obstruct most of the individuals who read the books.

All intellectual property rights are just licenses granted by society because it was thought, rightly or wrongly, that society as a whole would benefit by granting them. But in any particular situation, we have to ask: are we really better off granting such license? What kind of act are we licensing a person to do?

The case of programs today is very different from that of books a hundred years ago. The fact that the easiest way to copy a program is from one neighbor to another, the fact that a program has both source code and object code which are distinct, and the fact that a program is used rather than read and enjoyed, combine to create a situation in which a person who enforces a copyright is harming society as a whole both materially and spiritually; in which a person should not do so regardless of whether the law enables him to.

"Competition makes things get done better."

The paradigm of competition is a race: by rewarding the winner, we encourage everyone to run faster. When capitalism really works this way, it does a good job; but its defenders are wrong in assuming it always works this way. If the runners forget why the reward is offered and become intent on winning, no matter how, they may find other strategies—such as, attacking other runners. If the runners get into a fist fight, they will all finish late. Proprietary and secret software is the moral equivalent of runners in a fist fight. Sad to say, the only referee we've got does not seem to object to fights; he just regulates them ("For every ten yards you run, you are allowed one kick."). He really ought to break them up, and penalize runners for even trying to fight.

"Won't everyone stop programming without a monetary incentive?"

Actually, many people will program with absolutely no monetary incentive. Programming has an irresistible fascination for some people, usually the people who are best at it. There is no shortage of professional musicians who keep at it even though they have no hope of making a living that way.

But really this question, though commonly asked, is not appropriate to the situation. Pay for programmers will not disappear, only become less. So the right question is, will anyone program with a reduced monetary incentive? My experience shows that they will.

For more than ten years, many of the world's best programmers worked at the Artificial Intelligence Lab for far less money than they could have had anywhere else. They got many kinds of non-monetary rewards: fame and appreciation, for example. And creativity is also fun, a reward in itself.

Then most of them left when offered a chance to do the same interesting work for a lot of money.

What the facts show is that people will program for reasons other than riches; but if given a chance to make a lot of money as well, they will come to expect and demand it. Low-paying organizations do poorly in competition with high-paying ones, but they do not have to do badly if the high-paying ones are banned.

"We need the programmers desperately. If they demand that we stop helping our neighbors, we have to obey."

You're never so desperate that you have to obey this sort of demand. Remember: millions for defense, but not a cent for tribute!

"Programmers need to make a living somehow."

In the short run, this is true. However, there are plenty of ways that programmers could make a living without selling the right to use a program. This way is customary now because it brings programmers and businessmen the most money, not because it is the only way to make a living. It is easy to find other ways if you want to find them. Here are a number of examples.

A manufacturer introducing a new computer will pay for the porting of operating systems onto the new hardware.

The sale of teaching, hand-holding and maintenance services could also employ programmers.

People with new ideas could distribute programs as freeware, asking for donations from satisfied users, or selling hand-holding services. I have met people who are already working this way successfully.

Users with related needs can form users' groups, and pay dues. A group would contract with programming companies to write programs that the group's members would like to use.

All sorts of development can be funded with a Software Tax:

Suppose everyone who buys a computer has to pay x percent of the price as a software tax. The government gives this to an agency like the NSF to spend on software development.

But if the computer buyer makes a donation to software development himself, he can take a credit against the tax. He can donate to the project of his own choosing—often, chosen because he hopes to use the results when it is done. He can take a credit for any amount of donation up to the total tax he had to pay.

The total tax rate could be decided by a vote of the payers of the tax, weighted according to the amount they will be taxed on.

The consequences:

* the computer-using community supports software development.

* this community decides what level of support is needed.

* users who care which projects their share is spent on can choose this for themselves.

In the long run, making programs free is a step toward the post-scarcity world, where nobody will have to work very hard just to make a living. People will be free to devote themselves to activities that are fun, such as programming, after spending the necessary ten hours a week on required tasks such as legislation, family counseling, robot repair and asteroid prospecting. There will be no need to be able to make a living from programming.

We have already greatly reduced the amount of work that the whole society must do for its actual productivity, but only a little of this has translated itself into leisure for workers because much nonproductive activity is required to accompany productive activity. The main causes of this are bureaucracy and isometric struggles against competition. Free software will greatly reduce these drains in the area of software production. We must do this, in order for technical gains in productivity to translate into less work for us.

C.11 aixpdslib Warnings File

aixpdslib is an AIX archive machine maintained by the University of California Los Angeles. Several of the utilities on the accompanying CD were retrieved from the aixpdslib.seas.ucla.edu machine. They maintain a Warnings file with each system they distribute, which follows:

This software is provided AS IS, and neither the staff of SEASnet, The UCLA School of Engineering and Applied Science, The University of California at Los Angeles, the Regents of the University of California or any other person or entity is responsible for the condition of the programs in this archive. Nor are we responsible for the results of any attempt to copy, retrieve, compile, link or execute any program obtained from these archives. Nor are we responsible for the results if you follow any of the advice in any message, news item, or note contained in these archives.

In other words \fIcaveat emptor\fR, or in this case: Let the retriever beware.

Redundantly, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software.

NO WARRANTY

A. BECAUSE THE PROGRAM IS AVAILABLE FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

B. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The above two paragraphs are from the Free Software Foundation Copyright.

Appendix

D

Where to Go to Get More Information

D.1 Some Recommended Books

There are a wealth of books available in this market, but I thought it was important to point out a few of the better ones to get you started in the right direction. Some of the better are:

Any book written by Andrew Tanenbaum

Any Nutshell book (O'Reilly & Associates, Inc.)

Life with UNIX by Don Libes and Sandy Ressler

Any X Window System book by O'Reilly & Associates

The Matrix by John Quarterman

Any C programming books by Rex Jaeschke

UNIX System Security by Patrick Wood and Stephen Kochan

UNIX Network Programming by W. Richard Stevens

There are many others but you can't go wrong with those listed above.

D.2 Where You Can Get Help

Besides the free resources of the Internet including e-mail to experts and newsfeeds, you can purchase expertise. Lists of consultants exist in a variety of packages and areas on the Internet. See the file in the emacs distribution under the etc subdirectory entitled SERVICE. This has a listing of a variety of people who offer services and support contracts for free software, including most of the packages discussed in this book. If you are a commercial customer, this may not be a bad way to go.

Appendix

E

Internet Access Providers

The following is an approximate list of Internet service providers. To get a more current list, send e-mail to info-deli-server@netcom.com and put the text "send PDIAL" in the body of the message. You can also get a list by sending e-mail to dlist@ora.com.

TABLE E.1 Nationwide and International Service Providers

Provider	Coverage	Services
AARNet AARNet Support GPO Box 1142 Canberra ACT 2601 Australia +616 249 3385 +616 249 1369 (fax) aarnet@aarnet.edu.au	Australia	Dedicated (9.6KB–2MB) SLIP PPP
ANS (Advanced Networks and Services) 2901 Hubbard Road Ann Arbor, Ml 48105 (313) 663-7610 maloff nis.ans.net	Worldwide	Dedicated (1.5MB–45MB)
a2i Communications 1211 Park Avenue #202 San Jose, CA 95132 info@rahul.net	Continental U.S.	Dial-up
CLASS (Cooperative Library Agency for Systems and Services) 1415 Koll Circle, Suite 101 San Jose, CA 95112-4698 (800) 488-4559 (408) 453-0444	National	Dial-up (member libraries only)
Demon Internet Services Demon System Ltd. 42 Hendon Lane London N3 lTT England +44 81 349 0063 internet@demon.co.uk	UK	Dial-up SLIP PPP

TABLE E.1 Nationwide and International Service Providers (*Continued*)

Provider	Coverage	Services
EUnet EUnet Support +31 20 59 25 12 4 glenneu.net	Europe	
PACCOM University of Hawaii, ICS 2565 The Mall Honolulu, HI 96822 (808) 956-3499 torben)hawaii.edu	Pacific Rim countries	Dedicated (64KB–1.5MB)
PSI (Performance Systems International) 1180 Sunrise Valley Drive Suite 1100 Reston VA 22091 (703) 620-6651 (703) 629-4586 (fax) PSlLink info@psi.com	Worldwide	Dedicated (9.6KB–1.5MB) Dial-up SLIP PPP/UUI
SprintLink SprintInternational 13221 Woodland Park Drive Herndon, VA 22071 (703) 904-2156 mkisericml.icp.net	Worldwide	Dedicated (9.6 KB–1.5MB)
UKnet UKnet Support +44 227 475497 postmasteruknet.ac.uk	UK countries	Dedicated Dial-up UUCP
UUNET Suite 570 3110 Fairview Park Drive Falls Church, VA 22042 (703) 204-8000 (800) 4UU-NET3 info@uunet.uu.net	Worldwide	Dial-up SLIP PPP UUCP Dedicated (9.6 KB–1.5MB)
The Well 27 Gate Five Road Sausalito, CA 94965 (415) 332-4335 info@well.sf.ca.us	Access through X.25 and direct dial	Dial-up
The World Software Tool and Die 1330 Beacon Street Brookline, MA 02146 (617) 739-0202	U.S.	Dial-up

TABLE E.2 Regional Service Providers

Provider	Coverage	Services
AccessNB* Computer Science Department University of New Brunswick Fredericton, NB Canada E3B SA4	New Brunswick, Canada	
ARnet* Walter Neilson (403) 450-5188	Alberta, Canada	
BARRNET William Yundt Pine Hall Room 115 Stanford, CA 94305-4122 (415) 723-3104 gd.whyforsythe.stanford.edu	San Francisco, CA area International—Far East	Dedicated Dial-up SLIP PPP
BCnet BCnet Headquarters 419-6356 Agricultural Road Vancouver, BC Canada V6T lZ2 (604) 822-3932 BCnet'ubc.ca	British Columbia	Dedicated (2.4KB–1.5MB)
CERFnet P.O. Box 85608 San Diego, CA 92186-9784 (800) 876-2373 (619) 455-3990 helpcerf.net	Southern CA International (Korea, Mexico, Brazil)	Dedicated (14.4KB–1.5MB) Dial-up (local and 800) SLIP PPP
CICnet ITI Building 2901 Hubbard Drive, Pod G Ann Arbor, MI 48105 (313) 998-6103 infocic.net	Midwest U.S. (IL IA MN WI MI OH IN)	Dedicated (56KB–1.5MB)
Colorado Supernet CSM ComputerCenter Colorado School of Mines 1500 Illinois Golden, CO 80401 (303) 273-3471 (303) 273-3475 (fax) info'csn.org	Colorado	Dedicated (9.6KB–1.5MB) Dial-up SLIP PPP

TABLE E.2 Regional Service Providers (*Continued*)

Provider	Coverage	Services
CONCERT P.O. Box 12889 3021 Cornwallis Road Research Triangle Park, NC 27709 (919) 248-1404 jrrconcert.net	North Carolina	Dedicated (56KB–1.5MB) Dial-up SLIP PPP/UU
JVNCnet Sergio Heker 6 von Neuman Hall Princeton University Princeton, NJ 08544 (609) 258-2400 market jvnc.net	Northeastern U.S. International	Dedicated (19.2KB–1.5MB) Dial-up SLIP
Los Nettos Information Sciences Institute 4676 Admiralty Way Marina del Rey, CA 90292 (310) 822-1511 los-nettos-requestisi.edu	Los Angeleo, CA area	Dedicated (1.5MB)
MBnet* Gerry Miller (204) 474-8230	Manitoba, Canada	
Merit 2200 Bonisteel Boulevard Ann Arbor, MI 48109-2112 (313) 764-9430 jogdenmerit.edu	Michigan	
MIDnet 29 WESC University of Nebraska Lincoln, NE 68588 (402) 472-5032 dmfwestie.unl.edu	U.S. Plains States (NE OK AR SD IA , KA MO)	Dedicated (56KB–1.5MB)
MRNet (Minnesota Regional Network) 511 11th Avenue So, Box 212 Minneapolis, MN 55415 (612) 342-2570 (612) 344-1716 (fax)	Minnesota	Dedicated (56KB–1.5MB)
MSEN 628 Brooks Street Ann Arbor, MI 48103 (313) 998-4562 info@msen.com	Michigan	Dedicated (9.6KB–1.5MB) Dial-up SLIP PPP

TABLE E.2 Regional Service Providers (*Continued*)

Provider	Coverage	Services
NEARnet BBN Systems and Technologies 10 Moulton Street Cambridge, MA 02138 (617) 873-8730 nearnet-join@nic.near.net	Northeastern U.S. (ME NH VT CT RI MA)	Dedicated (9.6KB–10MB) SLIP PPP
Netcom Online Communication Services 4000 Moorepark Avenue #209 San Jose, CA 95117 (408) 544-8649 ruthann@netcom.com	California (6 locations in major cities)	Dial-up
netIllinois Joel Hartman Bradley University 1501 W. Bradley Avenue Peoria, IL 61625 (309) 677-3100 (309) 677-3092 (fax) joel@lbradley.edu	Illinois	Dedicated (9.6KB–1.5MB)
NevadaNet University of Nevada System Computing Services 4505 Maryland Parkway Las Vegas, NV 89154 (702) 739-3557	Nevada	Dedicated
NLnet* Wilf Bussey (709) 737-8329	Newfoundland, Labrador	
NorthWestNet 2435 233rd Place NE Redmond, WA 98053 (206) 562-3000 ehood@nwnet.net	Northwestern U.S. (OR WA WY AK ID, MT ND),	Dedicated (56KB–1.5MB)
NSTN* 900 Windmill Road, Suite 107 Dartmouth, Nova Scotia Canada B3B 137 (902) 468-NSTN parsons@hawk.nstn.ns.ca	Nova Scotia, Canada	Dedicated (9.6KB–56KB) SLIP Dial-up
NYSERNet 111 College Place, Room 3-211 Syracuse, NY 13244 (315) 443-4120 luckett@nysernet.org	New York State	Dedicated (9.6KB–1.5MB) SLIP PPP Dial-up

TABLE E.2 Regional Service Providers (*Continued*)

Provider	Coverage	Services
OARnet Ohio Supercomputer Center 1224 Kinnear Road Columbus, OH 43085 (614) 292-9248 alisonosc.edu	Ohio	Dedicated SLIP PPP
Onet 4 Bancroft Avenue Rm 116 University of Toronto Toronto, Ontario Canada M58 1A1 (416) 978-5058 eugenevm.utcs.utoronto.ca	Ontario, Canada	
PEINet* Jim Hancock (902) 566-0450	Prince Edward Island, Canada	
PREPnet 305 S. Craig, 2nd Floor Pittsburgh, PA 15213 (412) 268-7870 twb+andrew.cmu.edu	Pennsylvania (Dial-in from outside PA accepted),	Dedicated (9.6KB–1.5MB) SLIP PPP
PSCnet Pittsburgh Supercomputing Center 4400 5th Avenue Pittsburgh, PA 15213 (412) 268-4960 hastingspsc.edu	Eastern U.S.	Dedicated
RISQ* 3744 Jean Brillant Bureau 500 Montreal, Quebec Canada H3T 1P1 (514) 340-5700 turcotteclouso.crim.ca	Quebec	
SASK#net* Dean C. Jones (306) 966-4860	Saskatchewan	
Sesquinet Office of Networking and Computing Rice University Houston, TX 77251-1892 (713) 527-4988 farrellrice.edu	Texas Latin America	Dedicated (8.6KB–1.5MB) SLIP

TABLE E.2 Regional Service Providers (*Continued*)

Provider	Coverage	Services
SURAnet 1353 Computer Science Center 8400 Baltimore Boulevard College Park, MD 20740-2498 (301) 982-4600 info@sura.net	Southeastern U.S. Caribbean Islands	Dedicated (56KB–1.5MB)
THEnet Texas Higher Education Network Information Center Austin, TX 78712 (512) 471-2444 infonic.the.net	Texas Limited Mexico	Dedicated (1.5MB) Dial-up SLIP
VERnet Academic Computing Center Gilmer Hall University of Virginia Charlottesville, VA 22903 (804) 924-0616 jajvirginia.edu	Virginia	Dedicated Dial-up SLIP PPP
Westnet 601 S. Howes, 6th Floor South Colorado State University Fort Collins, CO 80523 (303) 491-7260 pburnsyuma.acns.colostate.edu	Western U.S. (AZ CO ID NM, UT WY)	Dedicated
WiscNet 1210 W. Dayton Street Madison, WI 53706 (608) 262-8874 dorlmacc.wisc.edu	Wisconsin	Dedicated (56KB–1.5MB) Limited Dial-up/SLIP PPP
WVnet* Harper Grimm (304) 293-5192 cc0110416)wvnvm.wvnet.edu	West Virginia	Dedicated SLIP PPP

*The information for these providers was not verified by press time.

Index

ABOUT THE AUTHOR

Keven E. Leininger is Managing Director at DevTech Associates, a consulting firm that specializes in the integration of UNIX technology into large corporations. He serves on the Board of Editors of *Open Computing* magazine and has contributed to a variety of publications including *Optiv* and *CIO Journal.* Mr. Leininger is also the author of the *UNIX Developer's Tool Kit* and *Solaris Developer's Tool Kit,* available from McGraw-Hill.

SOFTWARE AND INFORMATION LICENSE

The software and information on this diskette (collectively referred to as the "Product") are the property of The McGraw-Hill Companies, Inc. ("McGraw-Hill") and are protected by both United States copyright law and international copyright treaty provision. You must treat this Product just like a book, except that you may copy it into a computer to be used and you may make archival copies of the Products for the sole purpose of backing up our software and protecting your investment from loss.

By saying "just like a book," McGraw-Hill means, for example, that the Product may be used by any number of people and may be freely moved from one computer location to another, so long as there is no possibility of the Product (or any part of the Product) being used at one location or on one computer while it is being used at another. Just as a book cannot be read by two different people in two different places at the same time, neither can the Product be used by two different people in two different places at the same time (unless, of course, McGraw-Hill's rights are being violated).

McGraw-Hill reserves the right to alter or modify the contents of the Product at any time.

This agreement is effective until terminated. The Agreement will terminate automatically without notice if you fail to comply with any provisions of this Agreement. In the event of termination by reason of your breach, you will destroy or erase all copies of the Product installed on any computer system or made for backup purposes and shall expunge the Product from your data storage facilities.

LIMITED WARRANTY

McGraw-Hill warrants the physical diskette(s) enclosed herein to be free of defects in materials and workmanship for a period of sixty days from the purchase date. If McGraw-Hill receives written notification within the warranty period of defects in materials or workmanship, and such notification is determined by McGraw-Hill to be correct, McGraw-Hill will replace the defective diskette(s). Send request to:

Customer Service
McGraw-Hill
Gahanna Industrial Park
860 Taylor Station Road
Blacklick, OH 43004-9615

The entire and exclusive liability and remedy for breach of this Limited Warranty shall be limited to replacement of defective diskette(s) and shall not include or extend to any claim for or right to cover any other damages, including but not limited to, loss of profit, data, or use of the software, or special, incidental, or consequential damages or other similar claims, even if McGraw-Hill has been specifically advised as to the possibility of such damages. In no event will McGraw-Hill's liability for any damages to you or any other person ever exceed the lower of suggested list price or actual price paid for the license to use the Product, regardless of any form of the claim.

THE McGRAW-HILL COMPANIES, INC. SPECIFICALLY DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Specifically, McGraw-Hill makes no representation or warranty that the Product is fit for any particular purpose and any implied warranty of merchantability is limited to the sixty day duration of the Limited Warranty covering the physical diskette(s) only (and not the software or in-formation) and is otherwise expressly and specifically disclaimed.

This Limited Warranty gives you specific legal rights; you may have others which may vary from state to state. Some states do not allow the exclusion of incidental or consequential damages, or the limitation on how long an implied warranty lasts, so some of the above may not apply to you.

This Agreement constitutes the entire agreement between the parties relating to use of the Product. The terms of any purchase order shall have no effect on the terms of this Agreement. Failure of McGraw-Hill to insist at any time on strict compliance with this Agreement shall not constitute a waiver of any rights under this Agreement. This Agreement shall be construed and governed in accordance with the laws of New York. If any provision of this Agreement is held to be contrary to law, that provision will be enforced to the maximum extent permissible and the remaining provisions will remain in force and effect.

$49.95 U.S.A.

Workstation

The first AIX/6000 developer's tool kit complete with precompiled software on CD

Here is an indispensable guide for all application developers working in IBM's AIX/6000 workstation environment. Featuring a CD-ROM with precompiled software, this is the first reference to put the full power of IBM's version of UNIX at your disposal.

The *AIX/6000 Developer's Tool Kit* offers you a wealth of practical tools and tips for working with AIX 3.2.5 and AIX 4.1, and explains in detail all the differences between AIX and other versions of UNIX.

Included is complete, step-by-step coverage of the AIX software environment ... AIX devices ... AIX software packaging and distribution ... general tools ... software development tools ... the Internet ... output, format, and display tools ... communications tools ... editors ... and games.

The book contains all relevant source code, plus binaries for both AIX 3.2.5 and AIX 4.1. You'll also find numerous screen images that help illustrate the look and feel of particular tools and facilitate application development.

Whether you're presently using AIX to perform your work or actively making the migration to AIX, this comprehensive tool kit provides everything you need to function effectively in the dynamic AIX and RS/6000 environment.

About the Author

Kevin Leininger is Director of Reengineering at Dev/Tech Associates, a systems integration firm specializing in UNIX. He is also the author of the *UNIX Developer's Tool Kit* and the Solaris Developer's Tool Kit, as well as coauthor of *Mosaic and the New Internet,* all available from McGraw-Hill.

About the Series

The J. Ranade Workstation Series is McGraw-Hill's primary vehicle for providing workstation professionals with timely concepts, solutions, and applications. Jay Ranade is also Series Editor in Chief of more than 150 books in the J. Ranade IBM and DEC Series and Series Advisor to the McGraw-Hill Series on Computer C[illegible]munications.

Jay Ranade, S[illegible] and best-selling computer author, is a consultant and Assis[illegible] Merrill Lynch.

P/N 037679-4
PART OF
ISBN 0-07-911993-X
90000
9 780079 119933

Cover: Square One Design, Albuquerque, NM

McGraw-Hill
Serving the Need for Knowledge
1221 Avenue of the Americas
New York, NY 10020