Forward
TECHNOLOGY INCORPORATED

XENIX™ SYSTEM

SYSTEM REFERENCE

VOLUME 4

Catalog no. 9100
Part no. 91F00D

Document no. 8603d-100-00

CONTENTS

CHAPTER 1

Introduction

This volume of the XENIX Programmer's Manual contains material primarily about system level concepts, functions, and commands; while some of these topics may not be of immediate concern to the average user, everyone should become familiar with maintenance functions and the utilities available on the system. This is especially the case if there is no administrator designated for a system, or if for some reason the user must undertake a system task, such as recovering from a crash or restoring files from a backup tape.

The first section of the volume deals with the duties assigned to the role of system administrator. These duties include: establishing user directories, passwords, and accounts; maintaining system security and file structure integrity; performing periodic backups of the file system and restoring these files in the case of disaster. The XENIX tools pertinent to these functions are described and some final words of advice are offered.

The second section of the volume deals with some special system functions. One of these is writing device driver routines to allow XENIX to communicate with the I/O devices configured for its system. Another is establishing communication between two or more XENIX systems connected by telecommunication or hard link. Finally, the topic of system security is addressed.

The last section of the volume contains a summary of some system level programs, commands, and utilities not dealt with elsewhere in this manual. Once again, although the user may not find these of immediate use, he should become aware of their existence, as well as their location in the manual, for future reference. Also included here are the games available on the XENIX system.

CHAPTER 2

INTRODUCTION TO SYSTEM ADMINISTRATION

Unlike those users for whom the operating system is merely a tool for program development and applications, the system administrator must take responsibility for the health and welfare of the system as a whole. These tasks include adding new users, insuring that adequate disk space is available for all users at all times, making backups to protect against user and hardware errors, and giving training and support. In short, it is the system administrator's job to undertake all the maintenance functions necessary to maintain efficient system operation.

It is the intent of this document to provide the necessary background information and instill the confidence required for a system administrator (or site coordinator). It is intended to be read as an initial orientation to the system administrator's functions, as well as a reference for later use. This document is a supplement, not a substitute, for the XENIX Programmer's Introduction. It is strongly recommended that all the documentation be read thoroughly before any attempt is made to administer the total system.

2.1 OPERATING SYSTEM OVERVIEW

Before introducing the actual duties of the XENIX system administrator, it is helpful to examine the evolution of operating systems in general, and the XENIX system in particular.

2.1.1 A Brief History

Early computer users often had the sole use of an entire computer system, and thus they were able to manage its resources for their own convenience. But as the number of users grew, it became clear that some kind of formal resource management would be necessary to ensure the efficient, equitable distribution of a system's physical resources among all the users of a single system.

Even in a system where jobs could only be processed one at a time, a user had to be prevented from monopolizing the computer and its resources. System management of the central processor was required to limit the time consumed by user programs. It was also obvious that computer memory had to be managed in order to protect users from destroying each other's data or programs. When direct access secondary memory was available in sufficient quantity to make permanent data storage feasible, system management of I/O devices and secondary memory became still more critical. Operating systems were developed to meet these requirements for system management.

2.1.2 The Contents of an Operating System

An operating system consists of those program modules resident in a computer which control its physical resources, including:

1. Processors
2. Main storage
3. I/O devices
4. Secondary storage
5. Files

These modules resolve conflicts, attempt to maximize performance, and simplify the effective use of the system. They act as the interface between user programs and the

physical computer hardware.

2.1.3 Why Operating Systems Are Important

While executing, an operating system controls the entire operation of a computer. For example, when several user programs compete simultaneously for resources, the operating system determines in which order the users will have access, and for what period of time. In addition, an operating system prevents a computer system from crashing if a device becomes inoperable, and assigns ``paths'' to I/O devices when they are needed.

In present day operating systems, the command language serves as an interface to the operating system. This language is the essential tool of the system administrator. Some special features of the XENIX command language and of the XENIX file structure are detailed below; they are also dealt with in the Programmer's Introduction.

2.1.4 The XENIX Operating System

XENIX is derived from UNIX, an operating system which has been field tested for almost a decade, largely in university environments where it has demonstrated a capacity to withstand abuse and function under heavy workloads.

XENIX is a multi-user, multi-tasking operating system. It allows more than one user full and complete access to all of its resources on a time sharing basis. Each user has the illusion that he, and he alone, has absolute access to the computer's resources. XENIX satisfies all the needs of the modern computer user, while offering a wealth of supporting programs which aid its users in performing a wide range of tasks. of the operating system and the activities of the system administrator are described.

2.2 USERS, GROUPS, AND PROTECTIONS

In this section, the concepts of ``user'',``group'', and ``protection'' are introduced; these are concepts that should be understood thoroughly in order to properly administer the system.

2.2.1 Users

Generally, a user is an individual authorized to access the computer's resources. This authorization consists of a valid login name and, optionally, a unique password. The process of gaining access to the system is called ``logging in.'' It is the system administrator's responsibility to decide which individuals are authorized to use the computer, and to what extent.

In general, a user is one who logs in and proceeds to edit, compile, or perform any every day functions. The user's scope of access to the other files in the system is determined by the various protection settings for directories and files. The normal user is prohibited from accessing files and directories which do not have the necessary permissions set.

There is, however, a type of user who has unlimited access to the entire system: the ``super-user. '' As the name implies, the super-user has extraordinary capabilities, and is not restricted by any protection setting; the super-user can access any file in the system. Since the XENIX file protection mechanism does not apply to the super-user, a simple mistake or mistype by the super-user can cause massive damage to everyone else's programs and data, and possibly even bring down the entire system.

When you first log into the system, you are automatically positioned in your `home' directory as defined by an entry in the file etc/passwd. Then, from this directory, you can move to any directory or file in the system depending on its access and protection settings.

It is for this reason that the number of individuals who can assume super-user powers must be kept to an absolute minimum. Even those users who are given the super-user password must be careful to log in as the super-user only when necessary; this includes the system administrator.

Once logged in, a user has access to any directory or file in the system which allows the appropriate access. A user has full access rights to all files and directories which

extend from his home directory.

2.2.2 Groups

It is generally good policy for a user to restrict access to the files and directories he owns by setting the various permission bits associated with that file. However, there may be certain files and directories that need to be shared among members of a group, while still remaining restricted to everyone else.

The group affiliation is a facility that allows groups of users to share files while still restricting access to unaffiliated users; otherwise, the group id may be made the same as the user id for joint access.

2.2.3 Protection

Each file or directory created has a set of ``protection bits.'' There are a total of nine (9) bits, which are divided into three categories of three bits each. The three bits in each category are, in order, the read, write, and execute permissions (rwx). The three categories are: user, group, and other, where the ``user'' is generally the owner of the file, the ``group'' includes members of the same group, and ``other'' is everyone else. Thus, a file can have different protection or access permission depending on the way each of the bits are set in each category.

When you examine a long listing from the `l` command, the following holds true:

```
r = read permission, `-' denies it.
w = write permission, `-' denies it.
x = execute permission, `-' denies it.
```

For example,

```
-rwxrwxrwx 1 joe   32 Oct 19 10:00 example
```

means that everyone, owner, group, and other, has full access rights. In the following file, permissions allow the owner and the members of his group full access rights, but everyone else has only read permission to the file:

```
-rwxrwxr-- 1 joe   32 Oct 19 10:00 example
```

Here is a case where the file permits the owner read and write access, but everyone else (group and other) only read

access:

```
-rw-r--r-- 1 joe   32 Oct 19 10:00 example
```

In this example, the permissions allow only the owner to read or write the file:

```
-rw------- 1 joe   32 Oct 19 10:00 example
```

No one else is able to access this file for any reason, (except the super-user).

In this final example, the permissions allow execute status for the owner, members of the group, and all other members:

```
-rwxr-xr-x 1 joe   32 Oct 19 10:00 example
```

2.2.4 Protection and Directories

A directory is just like any other file except that no user, not even the super-user, can write on a directory. Directories provide the mapping between the names of files and the files themselves, and thus impose a structure on the file system as a whole.

Each directory created has a set of protection bits associated with it, but these protection bits have somewhat different meanings for directories than they do for files, as described below.

2.2.5 Search Permission

```
drwx--x--- 2 joe   32 Oct 19 10:00 book
```

In an ordinary file, the `x' bit signifies execution capability, but it is obviously impossible to execute a directory. The 'x' in this case means that the owner can search the contents of the directory for other directories or files. Search permission may be given for that particular user, group, or other, depending on how the protection bits are set. In this example, the owner has read, write, and execute permission on the file. Search permission is also given to members of the same group; however, no one else can search the directory named ``book''. If either the owner or group wishes to access this directory, there is no problem. However, if anyone else desires to access this directory, they will get the following message:

```
/usr/joe/book: bad directory
```

Of course, the directory book is not actually a ``bad directory''. It simply can not be accessed by those who are denied search permission. If a user has search permission and read or write permission, that user may change the contents of that directory.

2.2.6 Read Permission

```
drwxr-xr-- 2 joe      32  Oct  9 23:32  book
```

When an individual has read `r' permission for a directory, the user can read any file within that directory for which the read bit has been set.

In this example the owner is given read, write, and search access to all the files in this particular directory. The members of the same group are only given read access and search access. Any member of the same group as the user's can search this directory and read any file in it. However, if the directory contains a subdirectory, that subdirectory must also allow search permission. Everyone else has only search permission in this directory.

As in the previous example, the file is in the directory book:

```
-rw-r---r-- 2 joe     108 Oct 19 23:44  chapter1
```

The protection setting for chapter1 allows the owner both read and write permission, while the other members of the group and everyone else have only read permission. However, due to the protection setting on the parent directory book, only the owner can add (create) or delete (remove) it.

2.2.7 Write Permission

```
drwxrw---x 2 joe      32 Oct 20 09:32  book
```

Write permission on a directory allows the user to add, delete, remove, and rename any file within the specified directory.

2.2.8 Adding a New User: Things to Consider

When a new user is to be allowed to log in, the login name and, optionally, the password need to be entered in the appropriate file. This file is named /etc/passwd and can only be edited by the super-user. Each entry contains the following information:

- login name
- encrypted password
- numerical user id
- numerical group id
- initial working directory
- program to use as the shell

The initial working directory is the user root, or home, directory described earlier in this chapter. This is the user's position after a proper login. The last field is the program that is to be the user's shell, or command line interpreter. Below is a sample /etc/passwd file:

```
root:/H4Gql5HCW7uk:0:50:Super User:/:
daemon:x:1:50::/:
cron:x:1:50::/:
sys:qa.v0rOz90Hlc:2:50::/sys:
bin:Ob3cNJIVqpk2A:3:50::/bin:
joe:MWxG24o.l18fM:50:50:Joe Smith:/usr/joe:
```

Using the entry `joe' from the example above, we can see that the login name is `joe'. The password is encrypted here for security reasons; this is done automatically by the passwd command. When the password is assigned to `joe', it should be entered in normal alphanumeric text. Next in the /etc/passwd file is the user's numerical identification. This id is associated with each file that `joe' creates and thus owns. The next entry is the numerical identification of the group with which joe is affiliated. The entry `Joe Smith' is optional and serves to further identify this user. The final entry is joe's home directory, where he is positioned after a valid login. Since there is no entry in the field that defines the shell, the default shell is specified. The above is a typical entry in the /etc/passwd file for a user.

The following procedure creates an entry in the /etc/passwd file for any authorized user. It is a straightforward

procedure; just follow the directions:

```
# ed /etc/passwd     ( must be super-user )
$a                   ( append to end of file )
joe::10:1::/usr/joe:( new user's entry)
.                    ( end of current input )
w                    ( write it to the file )
q                    ( quit the editor )
# mkdir /usr/joe     ( created working directory )
nhange its owner to joe )
# login joe          ( test to see if it works )
```

Although the initial assignment of a password is optional, the new user should be encouraged to assign a password immediately after the initial login. Alternatively, the system administrator may assign a password or suggest that it be changed.

When any user, including the super-user, first logs on, a file called .profile runs automatically. The information in this file sets some terminal characteristics and initializes some pathname variables. The .profile file should be placed in the users' home directory (i.e., /usr/joe/.profile). Here is a sample .profile file:

```
stty erase '^h' kill '^u'          ( set terminal)
date                               ( write out date )
MAIL=/usr/spool/mail/joe           ( path to mailbox )
PATH=:/usr/joe/bin:/bin:/usr/bin:( search path )
```

The first entry sets the kill and backspace characters for the terminal. These are set by XENIX when it is first booted, and are different characters. The next line sets the variable ``MAIL'' to the pathname of the user's (joe's) mailbox so that joe may send and receive mail. Finally, the variable ``PATH'' lists names and order of the directories that are to be searched when joe wants to execute a command.

Each user can send and receive mail from and to other users. However, a unique mailbox must be created for each user. The procedure, similar to the creation of a working directory, is given below:

```
# cd /usr/spool/mail              ( change directories )
# >joe                            (create mailbox )
# chown joe /usr/spool/mail/joe( change owner )
```

2.2.9 Removing a User

Occasionally, it may be necessary to edit the password file and remove old entries. Removing the entry from etc/passwd removes the ``user'' from the system, any files belonging to that user still remain in the system. Therefore, it is often wise to save the old user's files on tape or diskette. The commands tar and cd write all of joe's files to a tape.

```
cd /usr/joe
tar cmv *
```

After saving all of joe's files on tape, his directory can now be removed.

```
rm -fr /usr/joe
```

2.3 THE XENIX FILE SYSTEM

In this section, the XENIX concept of file systems is discussed, and some file system structures and formats are examined. The system administrator's duties in maintaining these file systems are also introduced. This chapter is crucial; however, the relevance of each section depends on the individual system's configuration, particularly in respect to the size of the disks and how they are partitioned.

2.3.1 What a File System Is

Because the earliest computer systems had no file systems, disk storage space had to be managed manually by the programmer. For instance, the programmer might decide that a certain program would need, say, 400 blocks of disk space. He might then consult a notebook, locate 400 unused blocks, and code the numbers of those blocks directly into his program. This technique may have been adequate for the earliest machines, with only a few hundred blocks of storage, running perhaps a dozen tasks a day. Clearly it is inadequate for a modern system like XENIX, which supports dozens of users simultaneously, running tens of thousands of tasks per day.

XENIX, therefore, handles the burden of disk storage management for all users of the system, allocating disk space upon demand, keeping track of where on the disk the data is written and retrieving any part of it when given the "pathname" of the file. When the data file is no longer needed XENIX will, upon command, return the space it

occupied to the free pool. Thus, a disk device contains not only the files themselves, but also various items of information needed to locate and manage the files.

This information, along with the data files themselves, is called a ``file system''; it is critical to XENIX system management. The simplest XENIX system contains only one disk device and, except for a small section reserved for swapping, the entire disk will be set up as a single file system.

A disk may be used as a single storage area or it may be partitioned into several distinct areas. Each of these distinct partitions may be a file system. In general, an organized collection of files is referred to as a ``file system''. Each file system has its own set of identifying information about the files that belong to it, including the size and number of files and free blocks contained. The block of the file system which contains this identifying information is called the ``super-block''.

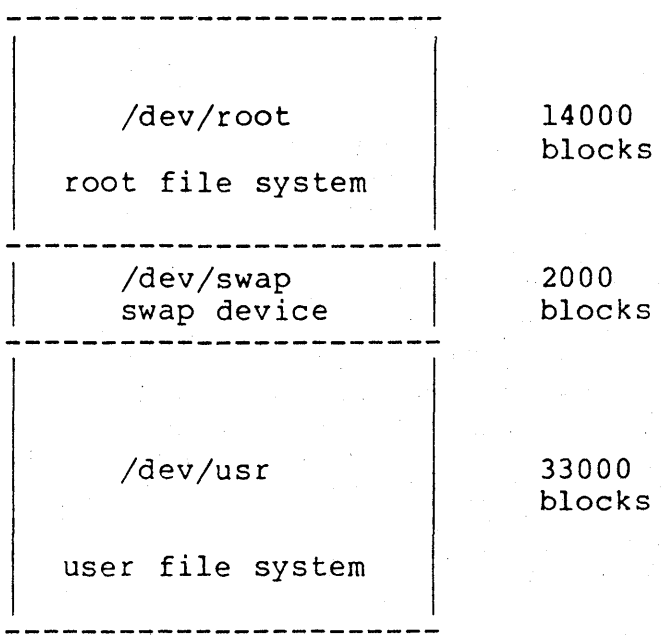
Each XENIX system has at least one disk drive containing either a fixed disk or a removable disk pack. These disks contain all the data in the system that is not actually being processed at any given moment, as well as the programs themselves. In the XENIX system, a file is simply a string of bytes. There is no logical record length and no particular record or file format imposed by the system.

2.3.2 A Simple File System: An Example

To XENIX, a file system is an organization of files which may occupy all or part of a disk. On a newly installed XENIX system there are two file systems resident on the disk. One of these is the root file system, where the operating system itself resides. The second is the user file system, consisting of user-created files. These file systems are distinct and logically separate from each other. Accordingly, each has a unique logical name. The name of the root file system is /dev/root; the name of the user file system is /dev/usr. If the name is preceded by an `r': /dev/rroot, /dev/rusr, the same XENIX file system is differently accessed. Some XENIX commands expect one name or the other, and will not operate if given the wrong one. Typically, when the prefix `r' is used, the command will run faster.

2.3.3 The Disk

Below is a diagram of a disk showing the relative size, in 512 byte blocks, of the root and user file systems. The relatively large section of blocks between them is not, strictly speaking, a file system; it is the system's ``swap'' device, a logical device resident on the disk which is used by XENIX to temporarily store images of the system's main memory during the execution of system processes. The swap device is distinct and logically separate from the user and root file systems. Accordingly, it has its own unique logical name: /dev/swap.



2.3.4 A Canonical File System

XENIX regards any disk, or part of a large disk, as a randomly addressable array of 512-byte blocks. These blocks are numbered consecutively 0, 1, 2, ... , on up to the size of the disk.

The first block (block 0) is unused by the file system, and is reserved for booting purposes. Block 1 (the second block) is the ``super-block'' which contains information about the file system. The third block contains the list of file definitions. The rest of the blocks are either occupied by file storage or remain free blocks.

The primary file system in XENIX is called the ``root'' file system. It contains the minimum data necessary to run the

system. It is always ``mounted'', or accessible.

```
ls -al /

drwxr-xr-x 2 bin      2704 Oct  7 08:32 bin
-r--r----- 1 root    11388 Apr 24 05:27 boot
-rwxr----- 1 root      374 Dec 10  1980 checkall
drwxr-xr-x 3 bin      2320 Oct  7 15:25 dev
drwxr-x--x 3 root     1088 Oct  5 14:02 etc
drwxr-x--- 9 root      464 Jul 25 16:54 fs
drwxr-x--x 2 root      592 Sep 29 01:03 lib
drwxr-x--- 2 root      256 May 19 09:37 lost+found
drwxrwxrwx 2 root       32 Jul 20 23:12 mnt
drwxr-x---11 root      288 Oct  3 15:17 stuff
drwxr-x---11 root      336 Jun 11 08:29 sys
drwxrwxrwx 7 root     1936 Oct  7 16:08 tmp
drwxr-xr-x75 root     1248 Sep 28 20:49 usr
drwxr-x--- 3 root       48 Dec 16  1980 v
-rwxr-xr-- 1 root    73368 Sep  8 15:22 xenix
```

As seen in the above example, all but three entries are directories; these files are the absolute minimum required to bring up XENIX. When XENIX is first booted, it is in single user mode; that is, it operates with the super-user, or root. When given the command to go multi-user, a shell program, /etc/rc is executed. One of the functions of this file it to make the other file systems accessible. All other file systems are extended from the root file system. In order to access other file systems, they must be made known to XENIX. This is done with the mount command.

2.3.5 Mounted File Systems

The root file system contains the bare essentials needed to bring XENIX up and running; if others are to use the system, access to the other file systems is given with the mount command. The mount commands for normally mounted file systems should be put into /etc/rc to ensure that they will be available when multi-user mode is entered.

2.3.6 The /etc/rc File

Here is what a sample /etc/rc file should look like:

```
PATH=/bin:/usr/bin
rm /etc/mtab
cat /dev/null >/etc/utmp

/etc/mount /dev/r5l /tmp    >/dev/console
if test $? = 2; then
                echo "cleaning /dev/rr5l"
                fsck -y -t /tmpfsck /dev/rr5l
                /etc/mount /dev/r5l /tmp
fi                          >/dev/console
/etc/mount /dev/usr /usr    >/dev/console
if test $? = 2; then
                echo "cleaning /dev/rusr"
                            >/dev/console
                fsck -y -t /tmp/fsck /dev/rusr
                            >/dev/console
                /etc/mount /dev/usr /usr
                            >/dev/console
fi
/etc/asktime </dev/console >/dev/console 2>&1
(
 /bin/mount /dev/stuff /stuff  1>/dev/console 2>&1 ) &

/etc/dmesg - >>/usr/adm/messages
date >/etc/reboot.date
chmod a+w /etc/mtab
/usr/lib/ex2.0preserve -
rm -f /usr/spool/lpd/lock; /usr/lib/lpd
rm -f /usr/tmp/* /usr/tmp/.,* /usr/tmp/.e*
rm -f /tmp/* /tmp/.,* /tmp/.e*
/etc/update
/etc/cron
/etc/accton /usr/adm/acct
```

This single file performs three types of tasks:

- the mounting of file systems
- housekeeping tasks
- initiation of Daemon processes

2.3.7 The File /etc/ttys

This file is also essential to bringing the system up from a single- to a multi-user mode. It controls whether or not a login process will run at each terminal and what baud rate(s) the system will use to communicate with it.

Each line of the file describes one terminal port. For example, the following /etc/ttys file contains entries for four devices, the system console and three user terminals:

```
14 console
lhtty00
Ihtty01
Ihtty02
```

The first character of a line in /etc/ttys tells the system whether or not to run a login process at the terminal attached to the port. If the character is an `1' then a login process will run at the terminal. If the character is a `0' then a login process will not run at the terminal.

The second character in each line in /etc/ttys tells the system what baud rate(s) to use when communicating with a terminal attached to the port. This may be a single fixed baud rate or a number of baud rates through which the system cycles before it finds the right one for a particular terminal.

2.3.8 The File /etc/motd

The file /etc/motd is the system's ``message of the day''. This text file is sent to the user's terminal after the login procedure, containing any announcement the system administrator wishes to make to all users.

2.3.9 Mounting Other File Systems

In addition to the root file system, which is always mounted first, there will be a standard set of file systems that will need to be mounted every time the system is booted. These will typically include:

- a user file system (referred to as usr), where all the users' directories reside
- a temporary file system for intermediate, temporary files created by compilations and assemblies
- others for specific needs

2.4 MAINTENANCE TASKS OF THE SYSTEM ADMINISTRATOR

Two of the system administrator's primary responsibilities are maintaining file system integrity and ensuring that adequate free disk space is available to the users. This section describes the tools provided by XENIX to perform these and other maintenance tasks. These commands clean up files and file systems, initiate system or job accounting programs, and determine disk space usage.

2.4.1 Daemon Processes

In addition to those programs initiated by the system administrator, there are ``daemon'' programs that run automatically as long as the system is up, periodically checking the system or performing system functions. As an example, the /etc/update program is a daemon which forces disk updates every thirty seconds. Another example is the lineprinter daemon, lpr. Check the Programmer's Introduction for descriptions of some other daemon programs.

2.4.2 The Importance of Disk Space

As users compile programs, edit files, or perform other tasks, they are competing for a valuable resource: free disk space. On a typical system, the potential for running out of free disk space is very high. When this occurs, no new files can be created, nor can any existing files expand.

To prevent this situation, the system administrator needs to estimate in advance the amount of space required for each file system when the system is first configured. If possible, a file system should contain approximately 15% free space, more if the file system fluctuates, less if it is relatively static.

XENIX offers some tools for determining the status of free space in a particular file system as well as some techniques for freeing space if there is a shortage. This section summarizes the use of these XENIX tools. For more detailed information, read the corresponding sections in the XENIX Programmer's Introduction

2.4.3 Checking for Disk Space

There are some XENIX commands that will aid you in determining the status of disk space on a file system. They are:

- df- disk free
- du- disk usage
- find- find files
- quot- summarize file ownership

Each of these is discussed below with examples.

2.4.4 The df Command

This command prints out the number of free blocks available in whatever file system is specified. If no file system is specified, the free space in all normally mounted file systems is printed.

```
$df
/dev/root 1195
/dev/usr 5962
```

This indicates that the root file system contains 1195 free blocks; the usr file system has 5962 free blocks.

2.4.5 The du Command

Du gives the number of blocks that are used by files in the specified directory and each of its subdirectories.

```
$du /tmp
29                              /tmp/nedtmp
1                               /tmp/henry
2568                            /tmp/jgl
3                               /tmp/susr
1                               /tmp/jerry/myfs
156                             /tmp/jerry
3513                            /tmp
```

The last line reports the total number of blocks used by that directory and its subdirectories.

2.4.6 The find Command

The find command is a powerful tool for finding files by size, date, owner, and date of last access. This helps the system administrator locate old files that the user has neglected to remove.

In the following example, the find command searches for all binary files named core produced by the system during a local core dump, which have not been accessed in the last seven days. In this case it is probably safe to remove these files, since users rarely re-access core files a week later.

```
find //e -name core - atime +7 -exec ls-al\;
```

2.4.7 The quot Command

This command reports the number of blocks currently owned by each user in the specified file system, revealing the largest consumers of disk space in a particular file system.

2.4.8 Other Tools

There are a few other tools available to the system administrator. For example, the file, etc/motd, which contains the ``message of the day'', can be edited to inform users that space is low and that old files should be deleted, or a personal message, using the mail command, may be used to remind the offender to remove old files. However, these techniques may not prove sufficiently persuasive.

2.4.9 File System Integrity

A file system consists of files, and these files, in turn, consist of blocks of bytes. If a block of information is bad, then the file, and potentially the entire file system, is compromised. A file system's integrity is compromised when it is internally inconsistent; it does not necessarily imply any physical damage to the disk. XENIX has some tools to check file systems and, if necessary, repair them.

The file system should be checked:

- Whenever the system is first brought up
- When it exhibits any abnormal behavior
- Daily, simply as a precaution

Some of the programs used to check the integrity of the file system are described below.

2.4.10 The fsck Program

Every time a file is created, modified, or removed, the XENIX system performs a series of file system updates. These updates yield a consistent file system.

When the system is first brought up in single user mode, a file consistency check program is sometimes run automatically. **Fsck** is a multi-pass file system check program. Each pass over the file system invokes a different phase of the fsck program. After the initial setup, fsck performs successive phases on each file system. It checks blocks and sizes, path-names, connectivity, reference counts, and the free block list; it also performs some cleanup.

Here is a sample output:

```
$ fsck /usr

Phase 1 - Check Blocks
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Free List
xxx files  xxx blocks  xxx free
```

PHASE 1: CHECK BLOCKS (AND SIZES)
This phase involves the internal file descriptors (inodes), reporting on errors resulting from examining file blocks for bad or duplicate blocks, after checking file size and format.

PHASE 2: CHECK PATH-NAMES
In this phase, directory entries pointing to erroneous file descriptors (inodes) reported in Phase 1, are removed.

PHASE 3: CHECK CONNECTIVITY
This phase checks directory connectivity from the results of Phase 2, reporting error conditions resulting from unreferenced directories.

PHASE 4: CHECK REFERENCE COUNTS
Phase 4 checks the link count information from Phases 2 and 3. It reports on error conditions resulting from unreferenced files, incorrect link counts for files, directories, or special files, unreferenced files and directories, bad and duplicate blocks in files and directories, and incorrect total free inode counts.

PHASE 5: CHECK FREE-LIST
This phase checks the free-block list, and reports errors resulting from bad blocks in the free-list, bad free block counts, duplicate blocks in the free-list, unused blocks from the file system not in the free block list, and incorrect total free block counts.

2.4.11 The dcheck Program

In XENIX, a ``link'' allows a single file to appear in multiple directories; linked files are indistinguishable from each other. Dcheck reads the directories in a file system, comparing the number of links for each file to the number of directory entries by which it is referenced. As links to a file are removed, the link count is decremented by one. When the last link to a file is removed, the blocks of data containing that file are released and added to the free-list for use by other files. Dcheck reports when the number of entries (files) and links to the files are either not equal or are both equal to zero.

If there are no errors, dcheck's output is simply the name of the disk on which the file system is stored:

```
$ dcheck /dev/rroot
/dev/rroot:
```

If there are inconsistencies, it will output the following table:

```
$dcheck /dev/rusr
/dev/rusr:
     entries  link cnt
15040                          10
17887                          01
```

The numbers 15040 and 17887 represent the internal file definitions (inodes) of the erroneous files. The first line indicates that file definition 15040 has been allocated, but has no links; it does not appear in any directory. The second line indicates a link to a nonexistent file.

As stated above, Dcheck reports instances when the number of files and links are either zero or are unequal. In this case, a file is allocated, but it does not appear in any directory. To correct this, remove the file definition from the file system with the following command:

```
clri device inode(s)
```

Here the device is the disk on which the file system is stored and the inode is the internal file descriptor. The inode number can be determined with:

```
ls -i
```

After removing all the necessary file descriptors, the file system must be updated in order to reclaim the blocks previously allocated to the file. This can be done by entering fsck with the -s option:

```
fsck -s file system
```

The number of links to a file may be greater than the number of times it is referenced in directories. Each time a link to that file is removed (i.e., the file itself is removed) the link count and the number of entries will be decremented by one, until the number of entries is zero; then the procedure described above can be followed.

The situation in which the number of entries is greater than the link count is more serious. The file has some links to it, but the internal file descriptor count is less than the number of actual links. As the links to a file are removed, the link count is decremented by one until it equals zero; then the file blocks are deallocated and returned to the free list. However, the file will appear valid in some directories, even though it has been deallocated, resulting in a reference to a nonexistent file.

Although dcheck reports on this problem, it is fsck which actually performs the correction. Phase 4 of fsck repairs the reference count of a file within a file system.

2.4.12 The icheck Program

Each of the 512-byte blocks in a file system must be accounted for once, as either in use by a file, as free, or bad. Icheck counts all the blocks in a file system.

Any block number which appears more than once is, of course, a duplicate block number; any block number which does not occur is referred to as ``missing''. Every block number in a file system must fall within the starting and ending block numbers (the range) of that file system. Icheck does this `range checking' on each block, and each block out of range is designated a bad block. In this way, each block of a file

system is accountable.

Below is an example of the **icheck** output:

```
$ icheck /dev/rroot

files 320 (r=210,d=70,b=10,c=30)
used 2443 (i=143,ii=9,iii=0)
free 1315
missing 0
```

This output is explained as follows:

- files - refers to the total number of files in the file system stored on the particular device. It is broken down by the type of file: regular(r), block(b), and character(c) devices.

- used - is the total number of blocks that are used by the file system. In XENIX, a file can grow to a maximum of 1,082,201,087 bytes. This is accomplished by having three levels of indirection. An internal file descriptor contains 13 disk addresses. The first 10 point directly to the first 10 blocks of a file. If the file is larger than 10 blocks, then the 11th address points to a block that contains the addresses of the next 128 blocks of the file. If the file is still larger than this, then the 12th block points to up to 128 blocks, each, in turn, pointing to 128 blocks of the file. Files yet larger use the 13th address for a ``triple indirect'' address. Thus, the total number of blocks used is broken down into the blocks used by each level of indirection.

- free - this is the number of blocks that are currently free, and therefore available for allocation for use by other files.

2.4.13 Error Conditions

There are two classes of errors which are reported by icheck: errors in the free-list and errors in the file descriptors. These are discussed below.

2.4.14 Errors in the Free List

The number of free blocks is the number of blocks which are available for allocation for use by other files. If blocks are missing, they are unavailable for allocation; This is

not a disastrous situation, unless the total number of missing blocks is unusually high. However, if blocks appear on the free list which have already been allocated, blocks already allocated to an existing file may be allocated to another file. Icheck reports on this error class as follows:

```
$ icheck /dev/rusr
/dev/rusr:
files   18571 (r=16878,d=1693,b=0,c=0)
used   215617 (i=4696,ii=210,iii=0,d=210501)
free    10498
missing   793
```

The first line is simply the number of blocks that are missing from that file system; this disk space will be lost. The second line reports on blocks found in the free-list which are duplicates of blocks appearing either in some file, or earlier in the free-list. The very last line reports the total number of such blocks. If the last two diagnostics appear, the situation is potentially dangerous. It can be repaired with the fsck -s option. The first entry is the block number in question; the next entry is the file (inode) to which that block is allocated.

2.4.15 Errors in the Internal File Descriptors

In XENIX, the file descriptor is called an ``inode'', an integer used to refer to a file. Errors in this class are reported with:

```
b*bad;inode=i#;class=[iclass]
b*dup;inode=i#;class=[iclass]
99794 dup; inode=14131, class=data (huge)
75013 dup; inode=14131, class=data (huge)
75366 dup; inode=14131, class=data (huge)
164422 dup; inode=14131, class=data (huge)
75327 dup; inode=14131, class=data (huge)
193376 dup; inode=14131, class=data (huge)
195902 dup; inode=14131, class=data (huge)
74993 dup; inode=14131, class=data (huge)
31826 dup; inode=15002, class=data (small)
files   18571 (r=16878,d=1693,b=0,c=0)
used   215617 (i=4696,ii=210,iii=0,d=210501)
free    10498
missing   793
```

In each of these cases, unfortunately, all the files involved must be removed. In the``b*dup'' case, the duplication is noted at the second occurrence of the block;

the first occurrence must be located. The following command lists all the inodes associated with the same block number. All will have to be removed.

```
icheck -b /dev/file system
```

2.5 BACKUPS

This chapter deals with backups of the file system(s). The system administrator must have a systematic plan for scheduling the time and frequency of backups, determining what level of comprehensiveness is required, and deciding where, and for how long, backup tapes should be stored. Suggestions are provided here, along with procedures for doing routine restores and recovering from disasters.

2.5.1 When to Take Backups

A backup of a compromised file system is worthless. Before performing a backup, file system integrity should be checked and restored, if necessary, using the tools described in the preceding section.

Preferably, the file system should be dismounted; at the very least, there should be little or no activity on the file system proper, to avoid the modification of a file, or files, while a backup is in progress. Backups should be scheduled so that they have the least possible impact on users.

Regular backups are insurance against partial or total system loss. Because the file system is critical to XENIX, it is suggested that a full and complete backup be done at least once a month; intermediate backups should be performed daily. Considerable flexibility exists for the level at which intermediate backups are performed.

2.5.2 A Full Backup

A full backup copies the entire file system to a secondary storage medium, usually a tape. Since a relatively small proportion of the files in a file system change frequently, full backups at regular intervals may be supplemented by intermediate, or incremental, backups. The full backup is called a``level 0 dump''. In case total disaster strikes, the file system is restored primarily from the level 0 dump, supplemented with files restored from intermediate backups, as necessary. A level 0 dump should be performed for each

file system at least once a month, and kept on site so that it is readily available if a file system needs to be rebuilt. On the other hand, the second most recent level 0 dump should be kept off site, so if something happens to the on site storage area, recovery is still possible.

As an alternative to using the dump program, a complete copy of a disk can be made either to another disk, or tape. This method will make a copy, but it does not update other files used by the dump program. The name of this program is dd. Dd copies the specified input file to the specified output; block size may be specified to take advantage of physical I/O capacity:

```
dd if=/dev/filesystem of=/dev/device-name
```

The following will completely copy the usr file system to a 800bpi tape

```
dd if=/dev/usr of=/dev/rmt0
```

To copy the file system to another disk:

```
dd if=/dev/usr of=/dev/device
```

However, the dump program is recommended for performing full backups.

2.5.3 Incremental Backups

An ``incremental'' backup copies only those files that have been changed after a given date, generally that of the last backup. There are ten different levels of dumps: 0-9, starting with the level 0 dump presented above, and passing through successive intermediate levels. Each time a successful backup is performed, the date of the backup is entered in the file /etc/ddate along with the name of the file system from which the backup was taken. All files modified since the most recent date stored in /etc/ddate for that file system at a lesser level will be copied.

For example, if a level 0 dump was taken of the usr file system on Oct 1, 1981, and on the next day, Oct 2, 1981, a level 9 backup was done, the files that would appear on the level 9 dump would be those that were modified between the level 0 dump and the level 9 dump. Or, using another example, assume that on Oct 24, 1981 a level 9 dump was performed, and on Oct 25, 1981 a level 5 dump was performed; The files appearing on the level 5 dump are those modified between the level 0 dump and the level 5 dump. On

subsequent level 9 backups, only those files that were modified since the last lower level backup (in this example, the level 5 backup) will be copied for that file system.

2.5.4 How to Perform a Backup

The Dump program copies the file system which resides on disk, to a secondary storage medium, usually a magnetic tape. The storage media should either be marked or cataloged so that there is a record of how much of a file system is backed-up, and where. As explained above, the file system to be backed-up should preferably be dismounted, or at least quiet. This is the command line which initiates the backup procedure.

dump level file-system

Dump is the name of the command; level is the level of the backup which is about to be done, and file system is the name of the file system which is about to be backed up.

dump 0u /dev/usr

This example initiates a full level 0 dump of the usr file system. The u argument is used to update the/etc/ddate file with the date of the dump. This file is used to determine which files are to be dumped for any intermediate incremental dump.

dump 7u /dev/usr

This command initiates a level 7 incremental backup of the usr file system. All files modified since the last date stored in/etc/ddate for the usr file system will be backed up.

2.5.5 Saving Backup Tapes

The system administrator should develop a consistent policy for the location and duration of tape storage. One possible approach is to save the full level 0 backup for an indefinite period after they are made, and the incremental backups for about four weeks.

It is very strongly recommended that the most recent backups, regardless of level, be stored on site for immediate use; the next most recent backups should be stored off site in a secure place, in case the on site backups are damaged or unusable.

2.5.6 Recovering From a Disaster

Backup tapes should be considered insurance against disaster, and hopefully, should rarely be used. Commonly, backup tapes are needed to restore files that are accidentally deleted or changed by users. The system administrator must often decide whether to restore lost files from backup tapes or attempt to repair the file system. Even with file systems that seem to be hopelessly corrupted, less time and data may be lost in repairing the file system than in attempting to recover an old version with a succession of incremental backup tapes.

The worst case is the total loss of an entire file system. To recover, the file system must be rebuilt from scratch, and as much information as possible restored.

Required for recovery are the most recent level 0 and intermediate level backups. By reading the backups in the correct order, the entire file system can often be restored to 98% of its normal state before it was destroyed. Restoring a file system is a long process; patience and a refusal to panic are essential. Backups are taken for just this type of emergency.

2.5.7 Restoration: Step 1

First, the file system needs to be reconstructed according to specifications. These specifications include the number of blocks in the file system and the size of the i-list. These parameters can be given with the command or placed in a file to be read by the system whenever the file system needs to be reconstructed. The example below constructs a file system called `usr'.

```
/etc/mkfs /dev/usr 9874 526
```

Its size will be 9874 blocks; the next number represents the size of the i-list in terms of inodes (an inode is 64 bytes; there are 8 i-nodes to a 512 byte block). This number represents the maximum number of files that the file system can hold. /etc/mkfs simply constructs an empty file system. The information from the backup tapes must still be restored.

The file system specifications can also be placed in a prototype file read by /etc/mkfs. This method is superior because it provides documentation for the file system in case it ever needs to be rebuilt. The following is a sample prototype file:

```
/sys/mdec/boot
31030 5024
d--755 3 1
$
```

The boot program is the first entry, on block 0 of the device upon which the file system resides. The next line, which specifies the size of the file system, is 31030 blocks in size; the i-list, in this case, is to be 5024 inodes (an inode is 64 bytes; there are 8 inodes to a 512 byte block). The next line represents the specifications of the root file for this file system. It consists of six characters: the first character represents the type of file, and skipping the next two characters, the rest of the string is a three digit octal number giving the owner, group, and other read, write, or execute permissions; the next two decimal numbers specify the user and group id of the owner of the file. Thus, the example above translates into:

```
/sys/mdec/uboot          (boot program)
31030 5024               (size of file system)
d-- 755 3 1              (specifications for root file)
$                        (end of input)
```

The system will construct the file system based on the information given to/etc/mkfs; this process takes approximately 15-20 minutes. Once the file system has been built, the next step, that of restoring the file, begins. Remember, that except for the root file system, all other file systems must be explicitly mounted on a directory which then becomes the root of that file system.

2.5.8 Restoration: Step 2

Now that an empty file system is reconstructed, the file system can be restored based on the information contained on the backup media. The level 0 backup must be done first, since it contains the entire file system; the incremental backups contain those files that have been modified subsequent to the level 0 backup. Mount the media and from the console, type:

```
# restor r /dev/usr
```

The system responds with:

```
# Last chance before scribbling on /dev/usr.
```

At this point, a skeleton file system resides on the device and the first volume of the level 0 dump is mounted. If

everything is ready, hit return. Reading the tape in this restoration process takes a while. After the first volume is read, the system prompts:

```
# Mount volume 2
```

It will continue to prompt for additional volumes until it reaches the end of the backup.

2.5.9 Fsck After the Level 0 Backup: Step 3

At this point, most of the file system has been restored. In order to insure that the file system is consistent, fsck and the other file system aids should be run. Once file system integrity is established, restore the information on the other incremental backups with:

```
# fsck /dev/usr
```

Fsck may ask you to supply a temporary scratch file; simply enter the name of a temporary file:

```
NEED SCRATCH FILE (212 BLOCKS)
ENTER FILENAME: /tmp/fsckaa
(this file should be on a file system that is,
or could be, accessible).
```

At this point, fsck will begin going through the five phases of checking the internal consistency of the file system.

2.6 SOME ADVICE FOR SYSTEM ADMINISTRATORS

Getting started as a system administrator is hard work, and there are no real shortcuts to a working knowledge of the system. You will need ample time for reading, study and hands-on experimenting. Don't commit yourself to ``going live'' with your system until you have had two weeks to teach yourself your job, and get the initial hardware quirks ironed-out.

Don't consign this guide to oblivion after initial system generation. In addition to needing it again whenever you add or change equipment, you will find that it contains valuable material about system tuning that appears nowhere else. As an administrator, you should be familiar with as much of the documentation as possible.

2.6.1 Disk Free Space

Making files is easy with XENIX. It has been said that the only standard thing about all XENIX systems is the message of the day telling users to clean up their files. If the free inode count falls below 100, the system spends most of its time rebuilding the free inode array. If a file system runs out of space, the system prints ``no-space'' messages and does little else. To avoid problems, the following free counts should be maintained:

- The file system containing /tmp (temporary files):
  - 16-user system: 1,500 free blocks.
  - 40-user system: 3,000 free blocks.
- The file system containing /usr:
  - 3,000 to 6,000 free blocks, depending on load.
- Other user file systems:
  - 6% to 10% free, depending on user habits (3,000 blocks minimum).

This brings up the associated question of how big a file system should be. Our preference is to set aside space on each drive for a copy of root/swap and use the rest of the pack for a single file system. However, if you have user groups that fight over disk space, it may be better to split them up arbitrarily (i.e., divide a pack into more than one file system).

Warning: if you set up different disk drives with differing cylinder partitions between file systems, it will probably

lead to an operations problem someday.

2.6.2 A Few Words About System Tuning

- File system reorganization, as described below, can help throughput, but at the expense of down time. It is helpful to undertake reorganization when the users are all asleep.

- If you use normal shutdown procedures, the file system check program, fsck, will help keep the disk free list in reasonable order.

- Try to keep disk drive usage balanced. If you have over 20 users, the root file system (/bin, /tmp, /etc, and swap) deserves a drive of its own.

- If you have a noisy modem (poorly executed do-it-yourself null-modem) or a disconnected modem cable, XENIX will spend a lot of CPU time trying to get it logged in. A random check of systems uncovers a lot of this going on.

2.6.3 Spare Disk Drive

- Without a spare disk drive, the system will be down when a drive is down.

- Without a spare drive, it is difficult to reorganize file systems or to restore user files.

2.6.4 Disk Packs

- Buy only fully ECC correctable packs and test them.

- If a pack develops uncorrectable errors, recondition it, or get rid of it.

2.6.5 Protecting User Files

Users, especially inexperienced ones, occasionally remove their own files. Open files are sometimes lost when the system crashes, and once in a great while, an entire file system will be destroyed (picture a disk controller that goes bad and writes when it should read). Here is a suggested file backup procedure:

- Each day, copy all user file systems to backup packs. Keep these packs 3 to 5 days before reusing them.

- Once a week, copy each file system to tape. Keep weekly tapes for 8 weeks.

- Keep bi-monthly tapes ``forever'' (they should be recopied once a year).

- The most recent weekly tapes should be kept off premises. The other tapes should be in a fire-proof safe, if you can afford one.

When XENIX goes down, active files can get scrambled. Your users will not want to start the day over each time your system fails. In addition to good backup, you must have file-system patching expertise available (on-site or on-call). If you ever re-boot the system for general use without checking out the file systems, disasters will occur. (in one case, five duplicate entries on a file-system free list ruined over 100 new files in just three days).

2.6.6 XENIX File System Backup Programs

The following backup programs are distributed:

- Dump/restor: This is a familiar tape-based system that has been used for several years. Full dumps are usually taken when the dump program warns that an incremental dump will run to more than one reel.

- Volcopy: physical file system copying to disk or tape. For those who can afford a spare drive, volcopy to disk provides convenient file restore and quick recovery from disk disasters (remember the spare drive). Tape volcopy provides good long-term backup because the file system can be read in and mounted quickly. Disk and tape volcopy are generally used together for short- and long-term backup. Volcopy can also be used for full dumps with either dump/restor or cpio/find.

We strongly recommend the spare disk drive; as explained above, the speed and convenience of volcopy are by no means the only advantage of a spare drive.

2.6.7 Controlling Disk Usage

If your XENIX system is a success, you will soon run out of disk space:

- During the considerable delay before you can get more drives, you will need to control usage:

  - Try to maintain the free space counts recommended above. Watch usage during the day by executing the df command regularly.

  - The du command should be executed after hours on a regular basis and the output kept in an accessible file for later comparison. In this way you can spot users who are rapidly increasing their disk usage.

  - The find can be used to locate inactive or large files. Example:

    find / -mtime +90 -atime +90 -print >somefile

    records in ``somefile'' the names of files neither written nor accessed in the last 90 days. Of course, this works best if you are super-user.

- You will also have to balance usage between file systems. To do this you will have to move user directories. Users should be taught to accept file system name changes (and to program around them, preferably ahead of time). The user's login directory name (available in the shell variable HOME) should be utilized to minimize path name dependencies. User groups with more extensive file system structures should set up a shell variable to refer to the file system name (e.g.: FS).

- The find and cpio commands can be used to move user directories and to manipulate the file system tree. The following sequence moves, via magnetic tape, the directory trees:

  user and usery from file system filesys1 to file system filesys2

  where more space is available:

```
cd /filesys1
find userx usery -cpio /dev/rmt0
cd /filesys2
mkdir userx usery
chown userx userx
chown usery usery
cpio -idmB </dev/rmt0
```

Make sure new copy is okay. Change userx and usery login directories in the /etc/passwd file:

```
rm -rf /filesys1/userx /filesys1/usery
```

When moving more than one user in this way:

- Keep users with common interests in the same file system (they may have linked files).
- Move groups of users who may have linked files with a single cpio (otherwise linked files will be unlinked and duplicated).

2.6.8 Reorganizing File Systems

The procedure for moving users described above can be expanded to provide a way to reorganize whole file systems. Reorganization can improve system response time. This is particularly true of the root file system which must be reorganized with all other file systems unmounted. Unfortunately, reorganization of a large file system is slow.

2.6.9 Keeping Directory Files Small

Directories larger than 5K bytes (320 entries) are very inefficient because of file system indirection. A user once complained that it took the system ten minutes to complete the login process; it turned out that his login directory was 25K bytes long, and the login program spent that time fruitlessly looking for a non-existent .profile file. A large /usr/mail or /usr/spool/uucp directory can also slow the system down. The following will locate such directories:

```
find / -type d -size +10 -print
```

Removing files from directories does not make the directories get smaller (the empty directory entries are available for reuse). The following will ``compact''

/usr/mail (or any other directory):

```
mv /usr/mail /usr/omail
mkdir /usr/mail
chmod 777 /usr/mail
cd /usr/omail
find . -print | cpio -plm ../mail
cd ..
rm -rf omail
```

2.6.10 Administrative Use of ``CRON''

The program cron is useful in the administration of the system; it can be used to:

- Turn off the programs in directory /usr/games during prime time.
- Run programs off-hours:
  - accounting
  - file system administration
  - long-running, user-written shell procedures (using the su command), for example:

```
su - userx userx_shell arg ...
```

2.6.11 Watch Out For Files and Directories That Grow

- /usr/adm/wtmp-login information;
- /usr/adm/pacct-process accounting; gets big quickly.
- /usr/lib/cronlog-status log of commands executed by cron(1M);
- /usr/spool-spooling directory for line printers, uucp, etc., and whose sub-directories should be compacted as described above.

2.6.12 Allocating Resources to Users

A prospective user should obtain connect-time and file-space authorization through appropriate channels. Once this is done, the user should apply for a login by providing the following information to the ``system administrator'':

- User's name.
- Suggested login name (not more than 8 characters, beginning with a lower-case letter).
- Relationships to other users (this influences the choice of the file system).
- Estimate of required file space (this also influences the choice of the file system).

Users should be forced to have passwords (not more than 8 characters long, but more than 5, and not in Webster's Unabridged);

2.6.13 Accounting and Usage

You should run the accounting programs even if you do not ``bill'' for service. Otherwise, your users' habits will be a mystery to you. Accounting information can also help you find performance bottlenecks, unused logins, bad phone lines, etc.

2.6.14 Line Printers

Most line printers are troublesome and impose considerable overhead on the system. Most also lack hardware tabs, character overstrike capability, etc. A printer that will work over an asynchronous link (DC1/DC3 protocol required) may be the best bet.

2.6.15 Security

The current XENIX is not tamper-proof. You can't keep people from ``breaking'' the system, but you can usually detect that they have done so. The following command will mail (to root) a list of all ``set user ID'' programs owned by root (super-user):

```
find / -user root -perm -4100 -exec ls -l {} ;
|mail root
```

Any surprises in root's mail should be investigated. Here is some related advice:

- Change the super-user password regularly. Don't pick obvious passwords (choose 6-to-8 character nonsense strings that combine alphabetics with digits or special

characters).

- If you have dial ports and do not require passwords, you are courting trouble.
- The chroot and su commands are inherently dangerous, as are group passwords; consider removing them from ``production'' systems.
- Login directories, .profile files, and files in /bin, /usr/bin, /lbin, and /etc that can be written to by other than their respective owners are security weak spots; police your system regularly against them.
- Remember, no time-sharing system with dial ports is really secure. Don't keep top-secret stuff on the system.

2.6.16 Communicating With Your Users

The directory /usr/news and the news command are provided as a way to get brief announcements to your users. More pressing items (one-liners) can be entered in the /etc/motd (message of the day) file; motd and (new to the user) news are announced at login time.

To reach users who are already logged in, use the wall (write all) command. Don't use wall while logged-in as super-user, except in emergencies.

The /usr/news directory should be cleaned out every few weeks so that nothing older than, say, three months is ever found there. The motd file should be cleaned out daily.

We have found that, on most systems, a file in /usr/news will reach 50% of users within a day and over 80% of users within a week.

2.6.17 Troubleshooting

It would be easy to write a book on this topic. The following are some of the key items involved in dealing with the hardware service contractor:

- Before you take out a hardware service contract, be sure that the contractor agrees to get along with the XENIX software (``It's the hardware,'' says you; ``It's the software,'' says the hardware service contractor).

- Keep on top of problems. Your contractor may have a problem-aging priority scheme; if so, make them prove that they are following it. Remember that an unreported problem is getting no priority at all. If a problem persists, escalate it up your contractor's local management chain; it may also be effective to complain to your contractor's sales representative.

- If you are serious about service to your users, you should have an extended-period service contract (e.g., 16 hours/day, 6 days/week). Arrange for preventive maintenance, non-critical repair, and add-on installation work to be done before or after prime time.

- If you have a service contract, learn the details. In particular, make sure that preventive maintenance is scheduled in advance and that it is completed.

- Ask the hardware service contractor to provide and maintain a ``site log''. You will have to work on the log as well.

- Make sure that your hardware vendor (as well as your hardware service contractor, if the two are different) agrees to the presence of other vendor's equipment on your system (even if you have none to start with).

- Run error logging. Keep console sheets. Make sure error messages are shown to your contractor's Customer Engineers.

- Take core dumps after system crashes and interpret results for Customer Engineers.

- Keep down-time records and make sure that your hardware service contractor knows about them.

Over 50% of your problems are likely to be related to the disk subsystem. As mentioned earlier, the way to keep your system up is to have a spare disk drive. Here are some key points to remember:

- Preventive maintenance of disk drives is very important.

- Make sure that the Customer Engineers who service your hardware see the error-logging printouts and console error messages produced by XENIX (and that they understand them).

- Disk failure can ruin a XENIX file system. The only defense is to make a complete, daily file backup!

Power supply modules are another common source of failure. Hard failure can be detected at the console; voltage drift is tougher.

CHAPTER 3

ADVANCED SYSTEM FUNCTIONS

In this section, tools which aid in the implementation of three advanced system functions are introduced: the writing of device driver routines, the establishment of inter-machine communication, and the maintenance of system security. Although the use of the XENIX tools described here may be limited to system administration, or some very specialized users, everyone should be familiar with their existence. In order to to assign I/O devices to a XENIX system, device driver routines are necessary; in order to establish either dial-up or hardwired communication between two or more XENIX systems, a series of UUCP programs are required. Detailed procedures for both of these functions are described below, followed by some words of advice concerning XENIX system security.

3.1 DEVICE DRIVER & I/O GUIDE

In order for the XENIX system to `talk' to a device, whether a tape, disk, terminal, or printer, a set of subroutines must be written and linked into the XENIX kernel. These subroutines are called device drivers, since they "drive" the devices they are written for. A device driver does two things: it interfaces the physical characteristics of a device to the operating system and it performs the low level data transfers that the operating system expects. Therefore, a device driver must be able to implement a set of standard I/O functions in a manner that is appropriate to a specified device. This section explains the XENIX I/O system, and provides the necessary information to write a device driver for a peripheral I/O device.

All devices have different characteristics; however, there are certain characteristics which are common to all devices. In addition to describing these common characteristics and giving an overview of the XENIX I/O system, this section presents prototype device drivers for the devices in the XENIX block I/O class. These drivers should be used only as models, and should not be copied as is.

3.1.1 The XENIX I/O System

The I/O system is broken into two separate systems or classes: the block I/O system and the character I/O system. The block I/O class is suitable for devices such as disks and tapes that work with addressable 512-byte blocks. Ordinary magnetic tapes just barely fit in this category, since by the use of forward and backward spacing any block can be read, even though blocks can only be written at the end of the tape. The block I/O class interface is very highly structured: these drivers share many routines, as well as a pool of buffers.

When a read or write takes place, the user's arguments and the file table entry are used to set up the following variables

```
u.u_base
u.u_count
u.u_offset
```

These contain the address of the I/O target area, the byte count for the transfer, and the current location in the file, respectively. If the file is a character-type special file, the appropriate read or write routine is called. It is responsible for transferring data and updating the count and

current location. Otherwise, the current location is used to calculate a logical block number in the file. If the file is an ordinary file, the logical block number must be mapped to a physical block number. All of this is done before the device driver is called.

3.1.2 Device Numbers

All XENIX devices are characterized by a major device number, a minor device number, and an I/O class (either block or character). These numbers are generally stored as an integer with the minor device number in the low-order 8 bits and the major device number in the next-higher 8 bits. The macros major and minor must be used to access these numbers. Both macros are found in the file /sys/h/param.h

XENIX uses the major device number to determine which driver will deal with which device. The minor device number is used internally by the driver at appropriate times, and is not used externally by the operating system.

Each I/O class has a table of entry points in /sys/conf/c.c for its device drivers. This configuration file is described in a later section. The major device number is used to index into the appropriate table for a particular device driver. The minor number is passed to the device driver as an argument. Typically, the minor number selects a subdevice attached to a given controller, or one of several similar hardware interfaces. It has no significance other than that attributed to it by the driver.

3.1.3 Block I/O System

A canonical block I/O device consists of randomly addressed, secondary memory blocks of 512 bytes each. The blocks are uniformly addressed from 0 up to the size of the device. The block device driver has the job of emulating this model on the physical device.

The block I/O devices are accessed through a layer of buffering software. The system maintains a list of buffers, each assigned a device name and a device address. This buffer pool constitutes a data cache for the block devices. On a read request, the cache is searched for the desired block. If the block is found, the data is made available to the requester without any physical I/O. If the block is not in the cache, the least recently used block in the cache is renamed, the correct device driver is called to fill up the renamed buffer, and then the data are made available.

Write requests are handled in a similiar manner. The correct buffer is found and relabeled if necessary. The write is performed simply by marking the buffer as `dirty'. The physical I/O is then deferred until the buffer is renamed.

3.1.4 Character I/O System

The character I/O system consists of all devices that do not fall into the block I/O model. This includes the "classical" character devices such as communication lines, paper tape, and line printers. It also includes magnetic tape and disks when they are not used in a stereotyped manner. For example, tape containing 80-byte records and disks that are copied a track at a time fall into this category. I/O requests are sent to the device driver essentially unaltered. The implementation of these requests is up to the device driver.

3.1.5 Configuration Tables

The file /sys/conf/c.c contains configuration tables for all XENIX devices. This table must be updated whenever a new device or a new device driver is added to the system. The major device numbers given in these tables for block and character devices are used as an index to their respective device tables. To add a new device to the configuration table, the major device number is selected by counting the line number (from zero) until the device is found, or a new entry (line) must be made. The minor device is the drive number, unit number, or partition. Each of the I/O classes requires a different interface, and therefore, different entries are made in the configuration table. The cdevsw table lists the interface routines which are present for a specific character device. Each line in the table specifies a device specific routine for open, close, read write, and special functions. If there is no open or close routine, `nulldev' may be given; if there is no read, write, or status routine, `nodev' may be given. Nodev sets an error flag and returns.

Each of the device driver categories is described below.

open - the open routine is called each time the file is opened. Its first argument is the full device number. The second argument is a flag which is nonzero only if the device is to be written upon.

```
devopen(dev,flag)
{
}
```

close - the close routine is called only when the file is closed for the last time; that is, when the very last process in which the file is open closes it. The first argument is the device number; the second is a flag which is nonzero if the file was open for writing in the process which performs the final close. Note that the flag does not indicate if the file has been written since the initial open.

```
devclose(dev,flag)
{
}
```

write - the write routine should copy the transfer count characters (u.u_count in /sys/h/user.h) from the buffer to the device, decrementing the count for each character passed. Successive calls on it return the characters to be written until the transfer count goes to zero(0) or an error occurs, in which case it a returns a negative one (-1).

```
devwrite(dev)
{
}
```

read - the read routine is called under conditions similar to write, except that the transfer count (u.u_count) is guaranteed to be nonzero.

```
devread(dev)
{
}
```

special-function - the `special-function' routine is invoked by the ioctl system call as follows:

```
devioctl(dev,cmd,cmarg,flag)
```

where dev is the device number, cmd is the command, and cmarg is a pointer to somewhere in user's memory, and flag is a copy of the flags associated with the file (see /sys/h/file.h).

Finally, each device should have appropriate interrupt-time routines. When an interrupt occurs, it is turned into a C-compatible call to the device's interrupt routine. After the

interrupt has been processed, a return from the interrupt handling routine returns from the interrupt itself.

When a device driver is running, it is often necessary to disable the processor's interrupt facility while critical sections of the interrupt code are being executed. This is done by changing the system priority level. The following functions are available for this purpose: spl4(), spl5(), spl6(), and spl7() These routines return a value which is suitable as an argument to splx(ps) which is used to restore the previous priority.

```
devintr(dev)
{
}
```

The bdevsw table contains the names of the interface routines and that of a table for each block device. As with character devices, block device drivers may supply an open and close routine called on each open and on the final close of the device. Instead of separate read and write routines, each block device driver has a strategy routine which is called with a pointer to a buffer header as an argument. The header contains a read/write flag, the core address, the block number, a byte count, and the major and minor device number. The role of the strategy routine is to carry out the operation as requested by the information in the buffer header, to sort the buffer headers (in cylinder order) for more efficient I/O, and then to call the device startup routine if the device is not already active. Although the most usual argument to the strategy routines is a genuine buffer header, all that is actually required is that the argument be a pointer to a place containing the appropriate information. The definition of a buffer header is given in /sys/h/buf.h..

The device's table specified by bdevsw has a byte to contain an active flag and an error count, a pair of links which constitute the head of the chain of buffers for the device (b forw, b back), and a first and last pointer for a device queue. Of these, all are used solely by the device driver itself except for the buffer-chain pointers. Since the buffers which have been handed over to the strategy routines are never on the list of free buffers, the pointers in the buffer which maintain the free list (av forw, av back), are also used to contain the pointers which maintain the device queues. The table used by the driver conventionally has the same format as a buffer header, with redefinitions of some of the names.

In addition to the routines contained in the drivers themselves, other system routines may be called and used by each driver. They are iodone, disksort, deverror and physio. Each of these will be briefly explained below.

The routine iodone arranges that the buffer to which points be released or awakened, as appropriate. This should be called when the driver has finished with the buffer, either normally or after an error.

Disksort (dp,bp) is called with two arguments: a pointer to the device and a pointer to the buffer header. Disksort sorts the buffer bp into the queue headed by dp. The I/O requests are sorted by cylinder number; the cylinder number must be computed before the call to disksort.

The routine deverror (bp, csr, stat) is called with three arguments: a pointer to the buffer header; the command register; the status register; The latter two arguments are used at the discretion of the driver. Deverror prints a diagnostic from a device driver. It prints the device, block number, and an octal word (usually some error status register) which were passed to it as arguments.

Physio (bp,&buf,dev,flag) is called with four arguments: a pointer to the buffer header, a buffer for raw I/O, a device, and a flag to indicate a read or write operation. Physio provides a means of using the raw, physical I/O to avoid the normal block buffering of the operating system, which improves transfer efficiency. The primary function of physio is to compute and validate physical addresses from the current logical address.

3.1.6 Writing the New Device Driver

To demonstrate how the new driver should be written, sources for the following two device drivers are presented:

bdproto	block type disk driver
btproto	block type tape driver

These psuedo-drivers are modeled from actual drivers on a PDP-11, and will differ from other device drivers primarily in their device register layout and the assignment of bits in the device registers. Each driver has a structure named `device' and a manifest named `XXADDR'. These define the device's register layout and its base address. To accommodate machines in which devices have a separate address space (unlike the PDP-11), all references to the device registers are through the subroutines in(addr) and

out(addr,val). To help identify areas in the driver that may be controller or device dependent, the comment `/* DEVICE DEPENDENT */' is used. When beginning to write a new device driver, it is best to begin with previously written routines such as those presented here. It is also necessary that the important information required to write a device driver be readily available. Thus, to describe the physical device to the system, it is necessary to have the following specifications:

1. Number of tracks
2. Number of sectors per track
3. Number of devices to be included
4. The device's address
5. The device's registers and their bit settings
6. The device's priority level

Sources for sample device drivers are given in Appendix A.

3.2 UUCP IMPLEMENTATION DESCRIPTION

Uucp is a series of programs designed to permit communication between XENIX systems using either dial-up or hardwired communication lines. It can be used for file transfers and remote command execution. This section describes the current implementation of the system.

Uucp is a batch operation. Files are created in a spool directory for processing by the uucp daemons. There are three types of files used for the execution of work:

- Data files contain data for transfer to remote systems.
- Work files contain directions for file transfers between systems.
- Execute files are scripts for commands that involve the resources of one or more systems.

There are four primary programs:

uucp builds work files and gathers data files in the spool directory for data transmission.

uux creates work files, execute files, and gathers data files for the remote execution of commands.

uucico executes the work files for data transmission.

uuxqt executes the scripts for XENIX command execution.

There are a couple of administrative programs:

uulog gathers temporary log files that may occur due to lockout of the uucp log file and reports some information such as copy requests and completion status.

uuclean removes old files from the spool directory.

The remainder of this section will describe the operation of each program, the installation of the system, the security aspects of the system, the files required for execution, and the administration of the system.

3.2.1 Uucp-XENIX to XENIX File Copy

The uucp command is the user's primary interface with the system. The command is designed to look like cp to the user. The syntax is

```
uucp
[
option
]
 ...   source ...   destination
```

where the source and destination may contain the prefix system-name!, which indicates the system where the file or files reside or where they will be copied.

Uucp has several options:

-d Make directories when necessary for copying the file.

-c Don't copy source files to the spool directory, but use the specified source when the actual transfer takes place.

-esys Send this job to system sys to execute. (Note that this will only work when the system sys allows uuxqt to execute a uucp command.)

-gletter Put letter in as the grade in the name of the work file. (This can be used to change the order of work for a particular machine.)

-m Send mail to the requester on completion of the work.

-nuser Notify user on the remote machine that a file has been sent.

There are several options available for debugging:

-r Queue the job but do not start uucico program.

-xnum is a level number between 1 and 9; higher numbers give more debugging output.

The destination may be a directory name, in which case the file name is taken from the last part of the source's name. If the directory exists, it must be writable by everybody. (Note that if the destination is a directory name and the ``-d'' option is specified to create the directory, the directory name must be followed by ``/''.) The source name may contain special shell characters such as ``?*[]''.

These will be expanded on the appropriate system.

The command

```
uucp   *.c   usg!/usr/dan
```

will set up the transfer of all files whose names end with ``.c'' to the ``/usr/dan'' directory on the``usg'' machine.

The source and/or destination names may also contain a user prefix. This translates to the login directory of user on the specified system. File names beginning with ~/'' translate into the public directory (usually /usr/spool/uucppublic) on the remote system. For names with partial path-names, the current directory is prepended to the file name. File names with ../ are not permitted for security reasons.

The command

```
uucp   usg! dan/*.h    dan
```

will set up the transfer of files whose names end with ``.h'' in dan's login directory on system ``usg'' to dan's local login directory.

For each source file, the program will check the source and destination file-names, the system-part of each argument, and the options to classify the work into several types:

1. Copy source to destination on local system.

2. Receive files from other systems.

3. Send files to a remote system.

4. Send files from remote systems to another remote system.

5. Receive files from remote systems when the source contains special shell characters as mentioned above.

6. Request that the uucp command be executed by a remote system.

After the work has been set up in the spool directory, the uucico program is started to try to contact the other machine and execute the work (unless the -r option was specified).

Type 1 - local copy The copy is done locally. The -m and -d options are not honored in this case.

Type 2 - receive files A workfile is created or appended with a one line entry for each request. The upper limit to the number of files per workfile is set in uucp.h. (The default setting is 20.) After the limit has been reached, a new work file is created. (All workfiles and executefiles use a blank as the field separator.) The fields for these entries are given below.

1. R
2. The full path-name of the source or a something/path-name. The something part will be expanded on the remote system.
3. The full path-name of the destination file. If the something notation is used, it will be immediately expanded.
4. The user's login name.
5. A ``-'' followed by an option list. The options -m and -d may appear.

Type 3 - send files Each source file is copied into a data file in the spool directory. (A ``-c'' option on the uucp command will prevent the datafile from being made. In this case, the file will be transmitted from the indicated source.) The fields for these entries are given below.

1. S
2. The full pathname of the source file.
3. The full pathname of the destination or something/file-name.
4. The user's login name.
5. A ``-'' followed by an option list. The options -d, -m, and -n may appear.
6. The name of the datafile in the spool directory. A dummy name, ``D.0'' is used when the -c option is specified.

7. The file mode bits of the source file in octal print format (e.g., 0666).

8. The user on the remote system to be notified upon completion of the file copy when the ``-n'' option is specified.

Type 4 and Type 5 - remote uucp required Uucp generates a uucp command and sends it to the remote machine; the remote uucico executes the uucp command.

Type 6 - remote execution This occurs when the ``-e'' option is used. In this case, the uux facility is used to create and send the request. This requires that the remote uuxqt program allows the uucp command.

3.2.2 Uux-XENIX To XENIX Execution

The uux command is used to set up the execution of a command where the execution machine and/or some of the files are remote. The syntax of the uux command is

```
uux
[
-
"] ["
option
 ]
 ... command-string
```

where the command-string is made up of one or more arguments. All special shell characters such as ``<>|^'' must be quoted either by quoting the entire command-string or quoting the character as a separate argument. Within the command-string, the command and file names may contain a system-name! prefix. All arguments that do not contain a ``!'' will not be treated as files. (They will not be copied to the execution machine.) An argument that contains a ``!'' but is not to be treated as a file at the present time, can be escaped by using ``()'' around the argument. (Note that the ``()'' symbols must usually be escaped with a ``\'' symbol.) The ``-'' is used to indicate that the standard input for command-string should be inherited from the standard input of the uux command. The following options are available for debugging:

-r Don't start uucico or uuxqt after queuing the job.

-xnum is a level number between 1 and 9; higher numbers give more debugging output.

The command

```
pr abc | uux - usg!lpr
```

will set up the output of ``pr abc'' as standard input to an lpr command to be executed on system ``usg''.

Uux generates an execute file that contains the names of the files required for execution (including standard input), the user's login name, the destination of the standard output, and the command to be executed. This file is either put in the spool directory for local execution or sent to the remote system using a send command (type 3 above).

For required files that are not on the execution machine, uux will generate receive command files (type 2 above). These command-files will be put on the execution machine for execution by the uucico program.

The execute file contains a script that will be processed by the uuxqt program. It is made up of several lines, each of which contains an identification character and one or more arguments. The lines are described below.

User Line

```
U user system
```

where the user and system are the requester's login name and system.

Required File Line

```
F file-name real-name
```

where the file-name is a unique name used for file transmission and real-name is the last part of the actual file name (contains no path information). Zero or more of these lines may be present. The uuxqt program will check for the existence of all these files before the command is executed.

Standard Input Line

```
I  file-name
```

The standard input is either specified by a ``<'' in the command-string or inherited from the standard input of the uux command if the ``-'' option is used. If a standard input is not specified, ``/dev/null'' is used. (Note that if there is a standard input specified, it will also appear in an ``F'' line.)

Standard Output Line

```
O  file-name  system-name
```

The standard output is specified by a ``>'' within the command-string. If a standard output is not specified, ``/dev/null'' is used. (Note that the use of ``>>'' is not implemented.)

Command Line

```
C  command
[
arguments
]
 ...
```

The arguments are those specified in the command-string. The standard input and standard output will not appear on this line. All required files will be moved to the execution directory (usually /usr/lib/uucp/.XQTDIR) and the XENIX command is executed using the shell specified in the uucp.h header file. In addition, a shell ``PATH'' statement is prepended to the command line as specified in the uuxqt program. (Note that a check is made to see that the command is allowed as specified in the uuxqt program.) After execution, the standard output is copied or sent to the proper place.

3.2.3 Uucico-Copy In, Copy Out

The uucico program will perform several major functions:

1. Scan the spool directory for work.
2. Place a call to a remote system.
3. Negotiate a line protocol to be used.

4. Execute all requests from both systems.

5. Log work requests and work completions.

Uucico may be started in several ways:

1. By a system daemon specified in a crontab entry,

2. By one of the uucp, uux, uuxqt or uucico programs,

3. directly by the user (this is usually for testing),

4. By a remote system. (The uucico program should be specified as the ``shell'' field in the ``/etc/passwd'' file for the logins used by remote systems to access uucp.)

When started by method 1, 2 or 3, the program is considered to be in MASTER mode. In this mode, a connection will be made to a remote system. If started by a remote system (method 4), the program is considered to be in SLAVE mode.

The MASTER mode will operate in one of two ways. If no system name is specified (-s option not specified) the program will scan the spool directory for systems to call. If a system name is specified, that system will be called, and work will only be done for that system.

Uucico is generally started by another program. There are several options used for execution:

-rl -Start the program in MASTER mode. This is used when uucico is started by a program or ``cron'' shell.

-ssys Do work only for system sys. If -s is specified, a call to the specified system will be made even if there is no work for system sys in the spool directory. This is useful for polling systems that do not have the hardware to initiate a connection.

The following options are used primarily for debugging:

-ddir Use directory dir for the spool directory.

-xnum Num is a level number between 1 and 9; higher numbers give more debugging output.

The next part of this section will describe the major steps within the uucico program.

Scan For Work The names of the work related files in the spool directory have the format

type . system-name grade number

where

1. type is an upper case letter (C - copy command file, D - data file, X - execute file),
2. system-name is the remote system,
3. grade is a character,
4. number is a four digit, zero padded sequence number.

The file C.res45n0031 would be a workfile for a file transfer between the local machine and the ``res45'' machine.

The scan for work is done by looking through the spool directory for work files (files with prefix ``C.''). A list is made of all systems to be called. Uucico will then call each system and process all work files.

Call Remote System The call is made using information from several files that reside in the uucp program directory (usually /usr/lib/uucp). At the start of the call process, a lock is set to forbid multiple conversations between the same two systems.

The L.sys file contains information required to make the remote connection:

1. System name
2. Times to call the system (days-of-week and times-of-day) and the minimum time delay before retry
3. Device or device type to be used for call
4. Line class (this is the line speed on almost all systems)
5. Phone number if field 3 is ACU or the device if not ACU
6. Login information (zero or more fields)

The time field is checked against the present time to see if the call should be made. The phonenumber may contain

abbreviations (e.g., mh, py, boston) that get translated into dial sequences using the L-dialcodes file.

The L-devices file is scanned using fields 3 and 4 from the L.sys file to find an available device for the call. The program will try each devices that satisfy 3 and 4 until a call is made, or no more devices can be tried. If a device is successfully opened, a lock file is created. If the call is completed, the login information (field 6 of L.sys) is used to login.

The conversation between the two uucico programs begins with a handshake started by the called, SLAVE, system. The SLAVE sends a message to let the MASTER know it is ready to receive the system identification and conversation sequence number. The response from the MASTER is verified by the SLAVE and if acceptable, protocol selection begins. The SLAVE can also reply with a ``call-back required'' message in which case, the current conversation is terminated.

Line Protocol Selection The remote system sends a message

P proto-list

where proto-list is a string of characters, each representing a line protocol.

The calling program checks proto-list for a letter corresponding to an available line protocol and returns a use-protocol message. The use-protocol message is

Ucode

where code is either a one character protocol letter or ``N'', which means there is no common protocol.

Work Processing The MASTER program does a work search similar to the one used in the ``Scan For Work'' section. (The MASTER has been specified by the ``-rl'' uucico option.) Each message used during the work processing is specified by the first character of the message:

S - send a file,

R -receive a file,

C -Copy complete,

X -execute a uucp command,

H -hangup.

The MASTER will send R, S or X messages until all work for the remote system is complete, at which point an H message will be sent. The SLAVE will reply with SY, SN, RY, RN, HY, HN, XY, or XN, corresponding to yes or no for each request.

The send and receive replies are based on permission to access the requested file/directory using the USERFILE and read/write permissions of the file/directory. After each file is copied into the spool directory of the receiving system, a copy-complete message is sent by the receiver of the file. The message CY will be sent if the file has successfully been moved from the spool directory to the destination. Otherwise, a CN message is sent. (In this case, the file is put in the public directory, usually /usr/spool/uucppublic, and the requester is notified by mail.) The requests and results are logged on both systems.

The hangup response is determined by a work scan of the SLAVE's spool directory. If work for the remote system exists an HN message is sent and the programs switch roles. If no work exists, an HY response is sent.

Conversation Termination When a HY message is received by the MASTER it is echoed back to the SLAVE and the protocols are turned off. Each program sends a final ``OO'' message to the other. The original SLAVE program will clean up and terminate. The MASTER will proceed to call other systems unless a ``-s'' option was specified.

3.2.4 Uuxqt-Uucp Command Execution

The uuxqt program is used to execute scripts generated by uux. The uuxqt program may be started by either the uucico or uux programs or a demon specified by a crontab entry. The program scans the spool directory for execute files (prefix ``X.''). Each one is checked to see if all the required files are available and if so, the command line is verified and executed.

The execute file is described in the ``Uux'' section above.

The execution is accomplished by executing a ``sh -c'' of the command line after appropriate standard input and standard output have been opened. If a standard output is specified, the program will create a send command or copy

the output file as appropriate.

Uulog-Uucp Log Inquiry When a uucp program can not make a log entry directly into the LOGFILE an individual log file is created: a file with prefix LOG. This will sometimes occur when more than one uucp process is running. Periodically, uulog may be executed to append these files to the LOGFILE.

The uulog program may also be used to request the output of LOGFILE entries. The request is specified by the use of the options:

-ssys Print entries where sys is the remote system name

-uuser Print entries for user user.

The intersection of lines satisfying the two options is output. A null sys or user means all system names or users respectively.

3.2.5 Uuclean-Uucp Spool Directory Cleanup

This program is typically started by the uucp daily demon. Its function is to remove files from the spool directory that are more than 3 days old. These are usually files for work that can not be completed. The requester of this work is notified that the files have been deleted.

There are several options:

1. -ddir The directory to be scanned is dir.
2. -m Send mail to the owner of each file being removed. (Note that most files put into the spool directory will be owned by the owner of the uucp programs since the setuid bit will be set on these programs. This mail is sometimes useful for administration.)
3. -nhours Change the aging time from 72 hours to hours hours.
4. -ppre Examine files with prefix pre for deletion. (Up to 10 of these options may be specified.)
5. -xnum This is the level of debugging output desired.

3.2.6 Security

The uucp system, left unrestricted, will let any outside user execute any commands and copy out/in any file that is readable/writable by a uucp login user. It is up to the individual sites to be aware of this and apply the protections that they feel are necessary.

There are several security features available aside from the normal file mode protections. These must be set up by the administrator of the uucp system.

1. The login for uucp does not get a standard shell. Instead, the uucico program is started so that all work is done through uucico.

2. The owner of the uucp programs should be an administrative login. It should not be one of the logins used for remote system access to uucp.

3. A path check is done on file names that are to be sent or received. The USERFILE supplies the information for these checks. The USERFILE can also be set up to require call-back for certain login-ids. (See the ``Files Required For Execution'' section for the file description.)

4. A conversation sequence count can be set up so that the called system can be more confident of the caller's identity.

5. The uuxqt program comes with a list of commands that it will execute. A ``PATH'' shell statement is prepended to the command line as specified in the uuxqt program. The installer may modify the list or remove the restrictions as desired.

6. The L.sys file should be owned by the uucp administrative login and have mode 0400 to protect the phone numbers and login information for remote sites.

7. The programs uucp, uucico, uux, uuxqt, uulog, and uuclean should be owned by the uucp administrative login, have the setuid bit set, and have only execute permissions.

3.2.7 Uucp Installation

It is assumed that the login name used by a remote computer to call into a local computer is not the same as the login name of a normal user or the uucp administrative login. However, several remote computers may use the same login name.

Each computer should be given a unique system name that is transmitted at the start of each call. This name identifies the calling machine to the called machine. The login/system names are used for security as described later in the USERFILE section.

There are several source modifications that may be required before the system programs are compiled. These relate to the directories, local system name, and attributes of the local environment.

There are several directories used by the uucp system:

1. lib (/usr/src/cmd/uucp) - This directory contains the uucp system source files.

2. program (/usr/lib/uucp) - This is the directory used for some of the executable system programs and the system files. Some of the programs reside in ``/usr/bin''.

3. spool (/usr/spool/uucp) - This is the uucp system spool directory.

4. xqtdir (/usr/lib/uucp/.XQTDIR) - This directory is used during execution of the uux scripts.

The names in parentheses above are the default values for the directories. The italicized names lib, program, xqtdir, and spool will be used in the following text to represent the appropriate directory names.

There are two files that may require modification, the makefile file and the uucp.h file. (On some systems, the makefile is named uucp.mk.) In addition, the ``uuxqt.c'' program may be modified as indicated in the ``Security'' section above. The following paragraphs describe the modifications.

Uucp.h modification Several manifests in ``uucp.h'' may need modification for the local system environment:

- UNAME should be defined if the ``uname'' function is available.
- MYNAME should be modified to the name of the local system if UNAME is not defined.
- ACULAST is the character required by the ACU as the last character. For most systems, it is a ``-''.
- DATAKIT should be defined if the system is on a datakit network.
- DIALOUT should be defined if the ``C'' library routine ``dialout'' is available.

Makefile Modification There are several make variable definitions that may need modification:

- INSDIR 10 is the program directory (e.g., INSDIR=/usr/lib/uucp). This parameter is used if ``make cp'' or ``make install'' is used.
- IOCTL is required to be set if the ``ioctl'' routine is not available in the standard ``C'' library; the statement ``IOCTL=ioctl.o'' is required in this case.
- PUBDIR is a public directory for remote access. This is also the login directory for remote uucp users. It should be the same as that defined in ``uucp.h''.
- SPOOL is the uucp spool directory. This should be the same as that defined in ``uucp.h''.
- XQTDIR is the directory for uuxqt to use for command execution. It is also defined in ``uucp.h''.
- OWNER is the administrative login for uucp.

Compile the System The command

```
make install
```

makes the required directories, compile all programs, set the proper file modes, and copy the programs to the proper directories. This command should be run as root. The command

```
make
```

compiles the entire system.

The programs uucp, uux, and uulog should be put in ``/usr/bin''. The programs uuxqt, uucico, and uuclean should be put in the program directory.

Files Required For Execution There are four files that are required for execution. They should reside in the program directory. The field separator for all files is a space.

L-devices This file contains call-unit device and hardwired connection information. The special device files are assumed to be in the /dev directory. The format for each entry is

```
type  line  call-unit  speed
```

where

1. type is a device type such as ACU or DIR. The field can also be used to specify particular ACUs for some calls by using a suffix on the ACU field, e.g., ACU3. This names should be used in L.sys.

2. line is the device for the line (e.g., cul0).

3. call-unit is the automatic call unit associated with line (e.g., cua0). Hardwired lines have a number ``0'' in this field.

4. speed is the line speed.

The line

```
ACU  cul0  cua0  300
```

would be set up for a system that has device ``/dev/cul0'' wired to a call-unit ``/dev/cua0'' for use at 300 baud.

L-dialcodes This file contains the dialcode abbreviations used in the L.sys file (e.g., py, mh, boston). The entry format is

```
abb  dial-seq
```

where abb is the abbreviation, dial-seq is the dial sequence to call that location. The line

```
py 165-
```

would be set up so that entry py7777 would send 165-7777 to the dial-unit.

USERFILE This file contains user accessibility information. It specifies four types of constraint:

1. which files can be accessed by a normal user of the local machine
2. which files can be accessed from a remote computer
3. which login name is used by a particular remote computer
4. whether a remote computer should be called back in order to confirm its identity.

Each line in the file has the format

```
login,sys
[
c
]
  path-name
[
path-name
]
 ...
```

where login is the login name for a user or the remote computer, sys is the system name for a remote computer, c is the optional call-back required flag, path-name is a path-name prefix that is acceptable for sys.

The constraints are implemented as follows.

1. When the program is obeying a command stored on the local machine, MASTER mode, the path-names allowed are those given on the first line in the USERFILE that has the login name of the user who entered the command. If no such line is found, the first line with a null login name is used.
2. When the program is responding to a command from a remote machine, SLAVE mode, the path-names allowed are those given on the first line in the file that has the system name that matches the remote machine. If no such line is found, the first one with a null system

name is used.

3. When a remote computer logs in, the login name that it uses must appear in the USERFILE. There may be several lines with the same login name but one of them must either have the name of the remote system or must contain a null system name.

4. If the line matched in 4 contains a ``c'', the remote machine is called back before any transactions take place.

The line

```
u,m  /usr/xyz
```

allows machine m to login with name u and request the transfer of files whose names start with ``/usr/xyz''.

The line

```
dan,  /usr/dan
```

allows the ordinary user dan to issue commands for files whose name starts with ``/usr/dan''. (Note that this type restriction is seldom used.)

The lines

```
u,m /usr/xyz  /usr/spool
u,  /usr/spool
```

allows any remote machine to login with name u. If its system name is not m, it can only ask to transfer files whose names start with ``/usr/spool''. If it is system m, it can send files from paths ``/usr/xyz'' as well as ``/usr/spool''.

The lines

```
root,  /
,   /usr
```

allow any user to transfer files beginning with ``/usr'' but the user with login root can transfer any file. (Note that any file that is to be transferred must be readable by anybody.)

Lsys Each entry in this file represents one system that can be called by the local uucp programs. More than one line

may be present for a particular system. In this case, the additional lines represent alternative communication paths that will be tried in sequential order. The fields are described below.

- system name

  The name of the remote system.

- time

  This is a string that indicates the days-of-week and times-of-day when the system should be called (e.g., MoTuTh0800-1730).

  The day portion may be a list containing some of

  Su Mo Tu We Th Fr Sa

  or it may be Wk for any week-day or Any for any day.

  The time should be a range of times (e.g., 0800-1230). If no time portion is specified, any time of day is assumed to be okay for the call. Note that a time range that spans 0000 is permitted, for example, 0800-0600 means all times are ok other than times between 6 and 8 am.

  An optional subfield is available to indicate the minimum time (minutes) before a retry following a failed attempt. The subfield separator is a ``,''. (e.g., Any,9 means call any time but wait at least 9 minutes after a failure has occurred.)

- device

  This is either ACU or the hardwired device to be used for the call. For the hardwired case, the last part of the special file name is used (e.g., tty0).

- class

  This is usually the line speed for the call (e.g., 300). The exception is when the ``C'' library routine ``dialout'' is available in which case this is the dialout class.

- phone

  The phone number is made up of an optional alphabetic abbreviation and a numeric part. The abbreviation

should be one that appears in the L-dialcodes file (e.g., mh5900, boston995-9980). For the hardwired devices, this field contains the same string as used for the device field.

- login

The login information is given as a series of fields and subfields in the format

```
[
expect  send
]
 ...
```

where expect is the string expected to be read and send is the string to be sent when the expect string is received.

The expect field may be made up of subfields of the form

```
expect[-send-expect] ...
```

where the send is sent if the prior expect is not successfully read and the expect following the send is the next expected string. (e.g., login--login will expect login; if it gets it, the program will go on to the next field; if it does not get login, it will send null followed by a new line, then expect login again.)

There are two special names available to be sent during the login sequence. The string EOT will send an EOT character and the string BREAK will try to send a BREAK character. (The BREAK character is simulated using line speed changes and null characters and may not work on all devices and/or systems.) A number from 1 to 9 may follow the BREAK for example, BREAK1 will send 1 null character instead of the default of 3. Note that BREAK1 usually works best for 300/1200 baud lines.

A typical entry in the L.sys file would be

```
sys Any ACU 300 mh7654 login uucp ssword: word
```

The expect algorithm match all or part of the input string as illustrated in the password field above.

3.2.8 Administration

This section indicates some events and files that must be administered for the uucp system. Some administration can be accomplished by shell files initiated by crontab entries. Others will require manual intervention. Some sample shell files are given toward the end of this section.

SQFILE - sequence check file This file is set up in the program directory and contains an entry for each remote system with which you agree to perform conversation sequence checks. The initial entry is just the system name of the remote system. The first conversation will add the conversation count and the date/time of the most recent conversation. These items will be updated with each conversation. If a sequence check fails, the entry will have to be adjusted manually.

TM - temporary data files These files are created in the spool directory while a file is being copied from a remote machine. Their names have the form

```
TM.pid.ddd
```

where pid is a process-id and ddd is a sequential three digit number starting at zero. After the entire file is received, the TM file is moved/copied to the requested destination. If processing is abnormally terminated the file will remain in the spool directory. The leftover files should be periodically removed; the uuclean program is useful in this regard. The command

```
program/uuclean -pTM
```

removes all TM files older than three days.

LOG - log entry files During execution, log information is appended to the LOGFILE. If this file is locked by another process, the log information is placed in individual log files which will have prefix LOG. These files should be combined into the LOGFILE by using the uulog program. This program appends the LOGFILE with the individual log files. The command

```
uulog
```

accomplishs the merge. Options are available to print some or all the log entries after the files are merged. The

LOGFILE should be removed periodically.

The LOG. files are created initially with mode 0222. If the program that creates the file terminates normally, it changes the mode to 0666. Aborted runs may leave the files with mode 0222 and the uulog program will not read or remove them. To remove them, either use rm, uuclean, or change the mode to 0666 and let uulog merge them into the LOGFILE.

STST - system status files These files are created in the spool directory by the uucico program. They contain information such as login, dialup or sequence check failures or will contain a TALKING status when two machines are conversing. The form of the file name is

STST.sys

where sys is the remote system name.

For ordinary failures, such as dialup or login, the file will prevent repeated tries for about 55 minutes. This is the default time; it can be changed on an individual system basis by a subfield of the time field in the L.sys file. For sequence check failures, the file must be removed before any future attempts to converse with that remote system.

LCK - lock files Lock files are created for each device in use (e.g., automatic calling unit) and each system conversing. This prevents duplicate conversations and multiple attempts to use the same device. The form of the lock file name is

LCK..str

where str is either a device or system name. The files may be left in the spool directory if runs abort (usually only on system crashes). They will be ignored (reused) after 1.5 hours. When runs abort and calls are desired before the time limit, the lock files should be removed.

ERRLOG - uucp system error file This file is created in the spool directory to record uucp system errors. Entries in this file should be rare. The messages come from the ASSERT statements in the various programs. Wrong modes on files or directories, missing files, and read/write system call failures on the transmission channel may cause entries in the ERRLOG file.

Shell Files The uucp program will spool work and attempt to start the uucico program, but uucico will not always be able to execute the request immediately. Therefore, the uucico program should be periodically started. The command to start uucico can be put in a ``shell'' file with a command to merge LOG. files and started by a crontab entry on an hourly basis. The file could contain the commands

```
/usr/bin/uulog
program/uucico
  -r1 -sinter
program/uucico
  -r1
```

The ``-r1'' option is required to start the uucico program in MASTER mode. The ``-s'' option can be used for polling as illustrated in the second line where machine inter is being polled. The third line will process all other spooled work.

Another shell file may be set up on a daily basis to remove TM, ST and LCK files and C. or D. files for work that can not be accomplished for reasons like bad phone number, login changes etc. A shell file containing commands like

```
program/uuclean
  -pTM -pC. -pD.
program/uuclean
  -pST -pLCK -n12
```

can be used. Note that the ``-n12'' option causes the ST and LCK files older than 12 hours to be deleted. The absence of the ``-n'' option will use a three day time limit.

A daily or weekly shell should also be created to remove or save old LOGFILEs. A shell like

```
cp
spool/LOGFILE

spool/o.LOGFILE
rm
spool/LOGFILE
```

can be used.

Login Entry Two or more logins should be set up for uucp. One should be an administrative login: the owner of all the uucp programs, directories and files. All others are used

by remote systems to access the uucp system. Each of the ``/etc/passwd'' entries for the access logins should have ``program/uucico'' as the shell to be executed. The login directory should be the public directory (usually /usr/spool/uucppublic). The various access login names are used in the USERFILE to restrict file access.

File Modes The programs uucp, uux, uucico, uulog, uuclean and uuxqt should be owned by the uucp administrative login with the ``setuid'' bit set and only execute permissions (e.g., mode 04111). The Lsys, SQFILE and the USERFILE, which are put in the program directory should be owned by the uucp administrative login and set with mode 0400. The mode of spool should be ``0755''. The mode of xqtdir should be ``0777''. The L-dialcodes and the L-devices files should have mode 0444.

3.3 XENIX SECURITY CONSIDERATIONS

Recently there has been much interest in the security aspects of operating systems and software. At issue is the ability to prevent undesired disclosure or destruction of information, and harm to the functioning of the system. This section discusses the degree of security which can be provided under the XENIX system and offers a number of hints on how to improve security.

3.3.1 Crashes and Slow-downs

XENIX like most other systems, was not developed with security in mind. The area of security in which XENIX is theoretically weakest is in protecting against crashing or at least crippling the operation of the system. The problem here is not mainly in uncritical acceptance of bad parameters to system calls- there may be bugs in this area, but none are known- but rather in lack of checks for excessive consumption of resources. Most notably, there is no limit on the amount of disk storage used, either in total space allocated or in the number of files or directories. Here is a particularly ghastly shell sequence guaranteed to stop the system:

```
while : ; do
        mkdir x
        cd x
done
```

Either a panic will occur because all the inodes on the device are used up, or all the disk blocks will be consumed, thus preventing anyone from writing files on the device.

In this version of the system, users are prevented from creating more than a set number of processes simultaneously, so unless users are in collusion it is unlikely that any one can stop the system altogether. However, creation of 20 or so CPU or disk-bound jobs leaves few resources available for others. Also, if many large jobs are run simultaneously, swap space may run out, causing a panic.

It should be evident that excessive consumption of disk space, files, swap space, and processes can easily occur accidentally in malfunctioning programs as well as at command level. In fact XENIX is essentially defenseless against this kind of abuse, nor is there any easy fix. The best that can be said is that it is generally fairly easy to detect what has happened when disaster strikes, to identify the user responsible, and take appropriate action. In

practice, we have found that difficulties in this area are rather rare, but we have not been faced with malicious users, and enjoy a fairly generous supply of resources which have served to cushion us against accidental overconsumption.

3.3.2 Protection and Permission

The picture is considerably brighter in the area of protection of information from unauthorized perusal and destruction. Here the degree of security seems nearly adequate theoretically, and the problems lie more in the necessity for care in the actual use of the system.

Each XENIX file has associated with it eleven bits of protection information together with a user identification number and a user-group identification number (UID and GID). Nine of the protection bits are used to specify independently permission to read, to write, and to execute the file to the user himself, to members of the user's group, and to all other users. Each process generated by or for a user has associated with it an effective UID and a real UID, and an effective and real GID.

When an attempt is made to access the file for reading, writing, or execution, the user process's effective UID is compared against the file's UID; if a match is obtained, access is granted provided the read, write, or execute bit for the user himself is present. If the UID for the file and for the process fail to match, but the GID's do match, the group bits are used; if the GID's do not match, the bits for other users are tested.

The last two bits of each file's protection information, called the set-UID and set-GID bits, are used only when the file is executed as a program. If, in this case, the set-UID bit is on for the file, the effective UID for the process is changed to the UID associated with the file; the change persists until the process terminates or until the UID changed again by another execution of a set-UID file.

Similarly the effective group ID of a process is changed to the GID associated with a file when that file is executed and has the set-GID bit set. The real UID and GID of a process do not change when any file is executed, but only as the result of a privileged system call.

The basic notion of the set-UID and set-GID bits is that one may write a program which is executable by others and which maintains files accessible to others only by that program.

The classical example is the game-playing program which maintains records of the scores of its players. The program itself has to read and write the score file, but no one but the game's sponsor can be allowed unrestricted access to the file lest they manipulate the game to their own advantage. The solution is to turn on the set-UID bit of the game program. When, and only when, it is invoked by players of the game, it may update the score file but ordinary programs executed by others cannot access the score.

There are a number of special cases involved in determining access permissions. Since executing a directory as a program is a meaningless operation, the execute-permission bit, for directories, is taken instead to mean permission to search the directory for a given file during the scanning of a path name; thus if a directory has execute permission but no read permission for a given user, he may access files with known names in the directory, but may not read (list) the entire contents of the directory. Write permission on a directory is interpreted to mean that the user may create and delete files in that directory; it is impossible for any user to write directly into any directory.

Another, and from the point of view of security, much more serious special case is that there is a ``super user'' who is able to read any file and write any non-directory. The super-user is also able to change the protection mode and the owner UID and GID of any file and to invoke privileged system calls. It must be recognized that the existence of a super-user is a potential threat to any protection scheme.

The first prerequisite for a secure system is arranging for all files and directories have the proper protection modes. Traditionally, XENIX software has been exceedingly permissive in this regard; essentially all commands create files readable and writable by everyone. In the current version, this policy may be easily adjusted to suit the needs of the installation or the individual user. Associated with each process and its descendants is a mask, which is in effect and-ed with the mode of every file and directory created by that process. In this way, users can arrange that, by default, all their files are no more accessible than they wish. The standard mask, set by login, allows all permissions to the user himself and to his group, but disallows writing by others.

To ensure both data privacy and integrity, it is usually sufficient, to make one's files inaccessible to others. A lack of sufficiency could result from the existence of set-UID programs created by the user and the possibility of a total breach of system security in ways such as those

discussed below. For greater protection, an encryption scheme is available. Since the editor is able to create encrypted documents, and the crypt command can be used to pipe such documents into the other text-processing programs, the length of time during which cleartext versions need be available is strictly limited. The encryption scheme used is not one of the strongest known, but it is judged adequate, in the sense that cryptanalysis is likely to require considerably more effort than more direct methods of reading the encrypted files. For example, a user who stores data that he regards as truly secret should be aware that he is implicitly trusting the system administrator not to install a version of the crypt command that stores every typed password in a file.

Needless to say, the system administrators must be at least as careful as their most demanding user to place the correct protection mode on the files under their control. In particular, it is necessary that special files be protected from writing, and probably reading, by ordinary users when they store sensitive files belonging to other users. It is easy to write programs that examine and change files by accessing the device on which the files are resident.

3.3.3 Password Security

On the issue of password security, XENIX is probably better than most systems. Passwords are stored in an encrypted form which, in the absence of serious attention from specialists in the field, appears reasonably secure, provided its limitations are understood. In the current version, it is based on a slightly defective version of the Federal DES; it has been purposely altered so that readily available hardware is useless for attempts at exhaustive key-search. Since both the encryption algorithm and the encrypted passwords are available, exhaustive enumeration of potential passwords is still feasible up to a point. We have observed that users choose passwords that are easy to guess: they are short, or from a limited alphabet, or in a dictionary. Passwords should be at least six characters long and randomly chosen from an alphabet which includes digits and special characters.

The set-UID (set-GID) notion must be used carefully if any security is to be maintained. The first thing to keep in mind is that a writable set-UID file can have another program copied onto it. For example, if the super-user (su) command is writable, anyone can copy the shell onto it and get a password-free version of su. A more subtle problem can come from set-UID programs which are not sufficiently

careful of what is fed into them. To take an obsolete example, the previous version of the mail command was set-UID and owned by the super-user. This version sent mail to the recipient's own directory. The notion was that one should be able to send mail to anyone even if they want to protect their directories from writing. The trouble was that mail was rather dumb: anyone could mail someone else's private file to himself. Much more serious is the following scenario: make a file with a line like one in the password file which allows one to log in as the super-user. Then make a link named ``.mail'' to the password file in some writable directory on the same device as the password file (say /tmp). Finally, mail the bogus login line to /tmp/.mail; you can then login as the super-user, clean up the incriminating evidence, and have your will.

3.3.4 Mounting Unauthorized Discs and Tapes

The fact that users can mount their own disks and tapes as file systems can be another way of gaining super-user status. Once a disk pack is mounted, the system believes what is on it. Thus one can take a blank disk pack, put on it anything desired, and mount it. There are obvious and unfortunate consequences. For example: a mounted disk with garbage on it will crash the system; one of the files on the mounted disk can easily be a password-free version of su; other files can be unprotected entries for special files. The only easy fix for this problem is to forbid the use of mount to unprivileged users. A partial solution, not so restrictive, would be to have the mount command examine the special file for bad data, set-UID programs owned by others, and accessible special files, and balk at unprivileged invokers.

CHAPTER 4

COMMAND REFERENCE

Included in this chapter are the XENIX Programmer's Manual manual pages for commands discussed in this manual. They have been included here for completeness.

NAME

ac - login accounting

SYNOPSIS

ac [-w wtmp **] [-p] [-d] [** people **] ...**

DESCRIPTION

Ac produces a printout giving connect time for each user who has logged in during the life of the current wtmp file. A total is also produced. **-w** is used to specify an alternate wtmp file. **-p** prints individual totals; without this option, only totals are printed. **-d** causes a printout for each midnight to midnight period. Any people will limit the printout to only the specified login names. If no wtmp file is given, /usr/adm/wtmp is used.

The accounting file /usr/adm/wtmp is maintained by init and login. Neither of these programs creates the file, so if it does not exist no connect-time accounting is done. To start accounting, it should be created with length 0. On the other hand if the file is left undisturbed it will grow without bound, so periodically any information desired should be collected and the file truncated.

FILES

/usr/adm/wtmp

SEE ALSO

init(8), login(1), utmp(5).

NAME

arcv - convert archives to new format

SYNOPSIS

arcv file ...

DESCRIPTION

Arcv converts archive files (see ar(1), ar(5)) from 6th edition to 7th edition format. The conversion is done in place, and the command refuses to alter a file not in old archive format.

Old archives are marked with a magic number of 0177555 at the start; new archives have 0177545.

FILES

/tmp/v*, temporary copy

SEE ALSO

ar(1), ar(5)

NAME

clri - clear i-node

SYNOPSIS

clri filesystem i-number ...

DESCRIPTION

Clri writes zeros on the i-nodes with the decimal i-numbers on the filesystem. After clri, any blocks in the affected file will show up as `missing' in an icheck(1) of the filesystem.

Read and write permission is required on the specified file system device. The i-node becomes allocatable.

The primary purpose of this routine is to remove a file which for some reason appears in no directory. If it is used to zap an i-node which does appear in a directory, care should be taken to track down the entry and remove it. Otherwise, when the i-node is reallocated to some new file, the old entry will still point to that file. At that point removing the old entry will destroy the new file. The new entry will again point to an unallocated i-node, so the whole cycle is likely to be repeated again and again.

SEE ALSO

icheck(1)

BUGS

If the file is open, clri is likely to be ineffective.

NAME

configure - generate new system configuration

SYNOPSIS

cd /sys/conf
configure [auto]

DESCRIPTION

Configure is used to create a new XENIX operating system. It interactively asks questions concerning the CPU and peripherals on the target computer in order to adjust certain internal parameters of the OS. After configure runs and the correct responses are made to the set of questions, the file xenix will be made in /sys/conf. This file can then be used for booting by moving it to the root. (It is wise to preserve the old xenix until the new one is proven.)

Configure is designed to be self-documenting and initially asks if the user wants information. Yes/no questions should be responded to with lines beginning with a y or n. Devices are always referred to by a two letter code; configure will list the appropriate codes for all devices it currently knows about. Numbers are decimal except when preceded by a 0 signifying octal radix.

Several files are produced by configure based on the responses to its questions. First, an assembly language support file is selected from /sys/conf/LIB0 and renamed to mch i.o or mch id.o depending on the particular target CPU. Next, numerous questions are asked concerning the peripheral devices, desired location of root, swap, and pipe file systems, internal parameter values, etc. The result is the file xenixconf. This file is then fed to the mkconf program which generates the files c.c and l.s. Finally, a make xenix i or make xenix id command is issued to perform the necessary assembly and linkage of the new executable operating system.

Except for the first execution, it is likely that some or all of the above files will remain valid when regenerating the system. Calling configure with the auto option causes the program to look for these files and bypass any questions and operations relating to the regeneration of files already present. This option is especially meaningful if the l.s file must be edited due to a non-standard configuration, or if the xenixconf file to be used has been set up by other than the last execution of configure. In general, once run without the auto option, the user may delete any of mch i.o, mch id.o, xenixconf, c.c, l.s, c.o, l.o, or xenix and then use configure auto to regenerate the missing data.

Release 2 of configure does not automate the handling of non-standard vectors, device addresses, and additional (user) device drivers. For now, the user must manually edit the l.s file and add any additional drivers to the /sys/dev/LIB2 archive to handle these situations. The auto option is useful under these circumstances.

FILES

mkconf
xenixconf
LIB0

SEE ALSO

mkconf(1m), `Setting up XENIX' in Volume 2

DIAGNOSTICS

Configure prints a set of messages and asks for the input again whenever the user types an unacceptable answer. Otherwise, the answers are fed, interactively line by line, to mkconf. The latter may also complain with sometimes cryptic messages. Since configure does not currently watch for complaints from mkconf, the user should probably hit the delete attention key, remove the xenixconf file and issue a configure auto command should an input error be reported by mkconf.

NAME

copy - copy groups of files

SYNTAX

copy [option] ... source ... dest

DESCRIPTION

The copy command copies the contents of directories to another directory. It is possible to copy whole file systems since directories are made when needed.

If files, directories, or special files do not exist at the destination, then they are created with the same modes and flags of the source. In addition, the super-user may set the user and group ids. The owner and mode will not be changed if the destination file exists. Note that there may be more than one source directory. If so, then the effect is the same as if the copy command had been issued, each with only one source.

All of the options must be given as separate arguments and they may appear in any order even after the other arguments. The arguments are:

-a Asks the user before attempting a copy. If the response does not begin with a 'y', then a copy will not be done. This option also sets the `-ad' flag.

-l Uses links instead whenever they can be used. Otherwise a copy is done. Note that links are never done for special files or directories.

-n Requires the destination file to be new. If not, then the copy command will not change the destination file. Of course the `-n' flag is meaningless for directories. For special files a `-n' flag is assumed (i.e., the destination of a special file must not exist).

-o Only the super user may set this option. If set then every file copied will have its owner and group set to those of the source. If not set, then the owner will be that of the user who invoked the program.

-m If set then every file copied will have its modification time and access time set to that of the source. If not set, then the modification time will be set to the time of the copy.

-r If set, then every directory is recursively examined as it is encountered. If not set then any

directories that are found will be ignored.

-ad Asks the user whether a `-r' flag applies when a directory is discovered. If the answer does not begin with a 'y', then the directory will be ignored.

-v If the verbose option is set, then all kinds of messages will be printed that reveal what the program is doing.

source This may be a file, directory or special file. It must exist. If it is not a directory, then the results of the command will be the same as for the cp command.

dest The destination must be either a file or directory different from the source.

If the source and destination are anything but directories, then copy will act just like a cp command. If both are directories, then copy will copy each file into the destination directory according to the flags that have been set.

DIAGNOSTICS

Should be self-explanatory

NAME

crypt - encode/decode

SYNTAX

crypt [password]

DESCRIPTION

Crypt reads from the standard input and writes on the standard output. The password is a key that selects a particular transformation. If no password is given, crypt demands a key from the terminal and turns off printing while the key is being typed in. Crypt encrypts and decrypts with the same key:

```
crypt key <clear >cypher
crypt key <cypher | pr
```

will print the clear.

Files encrypted by crypt are compatible with those treated by the editor ed in encryption mode.

The security of encrypted files depends on three factors: the fundamental method must be hard to solve; direct search of the key space must be infeasible; `sneak paths' by which keys or cleartext can become visible must be minimized.

Crypt implements a one-rotor machine designed along the lines of the German Enigma, but with a 256-element rotor. Methods of attack on such machines are known, but not widely; moreover the amount of work required is likely to be large.

The transformation of a key into the internal settings of the machine is deliberately designed to be expensive, i.e. to take a substantial fraction of a second to compute. However, if keys are restricted to (say) three lower-case letters, then encrypted files can be read by expending only a substantial fraction of five minutes of machine time.

Since the key is an argument to the crypt command, it is potentially visible to users executing ps(1) or a derivative. To minimize this possibility, crypt takes care to destroy any record of the key immediately upon entry. No doubt the choice of keys and key security are the most vulnerable aspect of crypt.

FILES

/dev/tty for typed key

SEE ALSO

ed(1), makekey(8)

NOTES

There is no warranty of merchantability nor any warranty of fitness for a particular purpose nor any other warranty, either express or implied, as to the accuracy of the enclosed materials or as to their suitability for any particular purpose. Accordingly, Bell Telephone Laboratories assumes no responsibility for their use by the recipient. Further, Bell Laboratories assumes no obligation to furnish any assistance of any kind whatsoever, or to furnish any additional information or documentation.

NAME

cu - call UNIX

SYNOPSIS

cu telno [-t] [-s speed] [-a acu] [-l line] [-nh]

DESCRIPTION

Cu calls up another XENIX system, a terminal, or possibly a non-XENIX system. It manages an interactive conversation with possible transfers of text files. Telno is the telephone number, with minus signs at appropriate places for delays, or `wait', to indicate a manual connection. If `wait' is specified, `/dev/null' is used as the dial unit and cu waits up to five minutes for the carrier to turn on. The -t flag is used to dial out to a terminal. Speed gives the transmission speed (110, 134, 150, 300, 600, 1200, 2400, 4800, 9600); 300 is the default value. The -nh flag prevents cu from hanging up the terminal line upon exit.

The -a and -l values may be used to specify pathnames for the ACU and communications line devices. They can be used to override the following built-in choices:

-a /dev/cua0 -l /dev/cul0

After making the connection, cu runs as two processes: the send process reads the standard input and passes most of it to the remote system; the receive process reads from the remote system and passes most data to the standard output. Lines beginning with `~' have special meanings.

The send process interprets the following:

~.	terminate the conversation.
~EOT	terminate the conversation
~<file	send the contents of file to the remote system, as though typed at the terminal.
~!	invoke an interactive shell on the local system.
~!cmd ...	run the command on the local system (via **sh -c**).
~$cmd ...	run the command locally and send its output to the remote system.
~%take from [to]	copy file `from' (on the remote system) to file `to' on the local system. If `to' is omitted, the `from' name is used both places.

~%put from [to]	copy file `from' (on local system) to file `to' on remote system. If `to' is omitted, the `from' name is used both places.
~%speed n	set speed of transmission line to `n', where n is one of 110, 134, 150, 300, 600, 1200, 2400, 4800, 9600.
~~...	send the line `~...'.

The receive process handles output diversions of the following form:

```
~>[>][:]file
zero or more lines to be written to file
~>
```

In any case, output is diverted (or appended, if `>>' used) to the file. If `:' is used, the diversion is silent, i.e., it is written only to the file. If `:' is omitted, output is written both to the file and to the standard output. The trailing `~>' terminates the diversion.

The use of ~%put requires stty and cat on the remote side. It also requires that the current erase and kill characters on the remote system be identical to the current ones on the local system. Backslashes are inserted at appropriate places.

The use of ~%take requires the existence of echo and tee on the remote system. Also, stty tabs mode is required on the remote system if tabs are to be copied without expansion.

FILES

/dev/cua0
/dev/cul0
/dev/null

SEE ALSO

dn(4), tty(4)

DIAGNOSTICS

Exit code is zero for normal exit, nonzero (various values) otherwise.

BUGS

The syntax is unique.

NAME

date - print and set the date

SYNTAX

date [yymmddhhmm [.ss]]

DESCRIPTION

If no argument is given, the current date and time are printed. If an argument is given, the current date is set. yy is the last two digits of the year; the first mm is the month number; dd is the day number in the month; hh is the hour number (24 hour system); the second mm is the minute number; .ss is optional and is the seconds. For example:

```
date 10080045
```

sets the date to Oct 8, 12:45 AM. The year, month and day may be omitted, the current values being the defaults. The system operates in GMT. Date takes care of the conversion to and from local standard and daylight time.

FILES

/usr/adm/wtmp to record time-setting

SEE ALSO

utmp(5)

DIAGNOSTICS

`No permission' if you aren't the super-user and you try to change the date; `bad conversion' if the date set is syntactically incorrect.

NAME

dcheck - file system directory consistency check

SYNOPSIS

dcheck [-i numbers] [filesystem]

DESCRIPTION

Dcheck reads the directories in a file system and compares the link-count in each i-node with the number of directory entries by which it is referenced. If the file system is not specified, a set of default file systems is checked.

The -i flag is followed by a list of i-numbers; when one of those i-numbers turns up in a directory, the number, the i-number of the directory, and the name of the entry are reported.

The program is fastest if the raw version of the special file is used, since the i-list is read in large chunks.

FILES

Default file systems vary with installation.

SEE ALSO

icheck(1), filsys(5), clri(1), ncheck(1)

DIAGNOSTICS

When a file turns up for which the link-count and the number of directory entries disagree, the relevant facts are reported. Allocated files which have 0 link-count and no entries are also listed. The only dangerous situation occurs when there are more entries than links; if entries are removed, so the link-count drops to 0, the remaining entries point to thin air. They should be removed. When there are more links than entries, or there is an allocated file with neither links nor entries, some disk space may be lost but the situation will not degenerate.

BUGS

Since dcheck is inherently two-pass in nature, extraneous diagnostics may be produced if applied to active file systems.

NAME

dd - convert and copy a file

SYNTAX

dd [option=value] ...

DESCRIPTION

Dd copies the specified input file to the specified output with possible conversions. The standard input and output are used by default. The input and output block size may be specified to take advantage of raw physical I/O.

option	values
if=	input file name; standard input is default
of=	output file name; standard output is default
ibs=n	input block size n bytes (default 512)
obs=n	output block size (default 512)
bs=n	set both input and output block size, superseding ibs and obs; also, if no conversion is specified, it is particularly efficient since no copy need be done
cbs=n	conversion buffer size
skip=n	skip n input records before starting copy
files=n	copy n files from (tape) input
seek=n	seek n records from beginning of output file before copying
count=n	copy only n input records
conv=ascii	convert EBCDIC to ASCII
ebcdic	convert ASCII to EBCDIC
ibm	slightly different map of ASCII to EBCDIC
lcase	map alphabetics to lower case
ucase	map alphabetics to upper case
swab	swap every pair of bytes
noerror	do not stop processing on an error
sync	pad every input record to ibs
... , ...	several comma-separated conversions

Where sizes are specified, a number of bytes is expected. A number may end with **k**, **b** or **w** to specify multiplication by 1024, 512, or 2 respectively; a pair of numbers may be separated by **x** to indicate a product.

Cbs is used only if ascii or ebcdic conversion is specified. In the former case cbs characters are placed into the conversion buffer, converted to ASCII, and trailing blanks trimmed and new-line added before sending the line to the output. In the latter case ASCII characters are read into the conversion buffer, converted to EBCDIC, and blanks added to make up an output record of size cbs.

After completion, dd reports the number of whole and partial input and output blocks.

For example, to read an EBCDIC tape blocked ten 80-byte EBCDIC card images per record into the ASCII file x:

```
dd if=/dev/rmt0 of=x ibs=800 cbs=80 conv=ascii,lcase
```

Note the use of raw magtape. Dd is especially suited to I/O on the raw physical devices because it allows reading and writing in arbitrary record sizes.

To skip over a file before copying from magnetic tape do

```
(dd of=/dev/null; dd of=x) </dev/rmt0
```

SEE ALSO

cp(1), tr(1)

DIAGNOSTICS

f+p records in(out): numbers of full and partial records read(written)

NOTES

The ASCII/EBCDIC conversion tables are taken from the 256 character standard in the CACM Nov, 1968. The `ibm' conversion, while less blessed as a standard, corresponds better to certain IBM print train conventions. There is no universal solution.

Newlines are inserted only on conversion to ASCII; padding is done only on conversion to EBCDIC. These should be separate options.

NAME

df - disk free

SYNOPSIS

df [filesystem] ...

DESCRIPTION

Df prints out the number of free blocks available on the filesystems. If no file system is specified, the free space on all of the normally mounted file systems is printed.

FILES

Default file systems vary with installation.

SEE ALSO

icheck(1)

NAME

du - summarize disk usage

SYNTAX

du [-s] [-a] [name ...]

DESCRIPTION

Du gives the number of blocks contained in all files and (recursively) directories within each specified directory or file name. If name is missing, '.' is used.

The optional argument -s causes only the grand total to be given. The optional argument -a causes an entry to be generated for each file. Absence of either causes an entry to be generated for each directory only.

A file which has two links to it is only counted once.

NOTES

Non-directories given as arguments (not under -a option) are not listed.
If there are too many distinct linked files, du counts the excess files multiply.

NAME

dump - incremental file system dump

SYNOPSIS

dump [key [argument ...] filesystem]

DESCRIPTION

Dump copies to magnetic tape all files changed after a certain date in the filesystem. The key specifies the date and other options about the dump. The key consists of characters from the set 0123456789fusd.

f Place the dump on the next argument file instead of the tape.

u If the dump completes successfully, write the date of the beginning of the dump on file `/etc/ddate'. This file records a separate date for each filesystem and each dump level.

0-9 This number is the `dump level'. All files modified since the last date stored in the file `/etc/ddate' for the same filesystem at lesser levels will be dumped. If no date is determined by the level, the beginning of time is assumed; thus the option 0 causes the entire filesystem to be dumped.

s The size of the dump tape is specified in feet. The number of feet is taken from the next argument. When the specified size is reached, the dump will wait for reels to be changed. The default size is 2300 feet.

d The density of the tape, expressed in BPI, is taken from the next argument. This is used in calculating the amount of tape used per write. The default is 1600.

If no arguments are given, the key is assumed to be 9u and the program attempts to dump the default filesystem to the default tape.

Now a short suggestion on how perform dumps. Start with a full level 0 dump

```
dump 0u
```

Next, periodic level 9 dumps should be made on an exponential progression of tapes. (Sometimes called Tower of Hanoi - 1 2 1 3 1 2 1 4 ... tape 1 used every other time, tape 2 used every fourth, tape 3 used every eighth, etc.)

```
dump 9u
```

When the level 9 incremental approaches a full tape (about 78000 blocks at 1600 BPI blocked 20), a level 1 dump should be made.

```
dump 1u
```

After this, the exponential series should progress as uninterrupted. These level 9 dumps are based on the level 1 dump which is based on the level 0 full dump. This progression of levels of dump can be carried as far as desired.

FILES

Default filesystem and tape vary with installation. For safety, however, we recommend that default disk filesystems not be used, as common operator errors can destroy that default disk.
/etc/ddate: record dump dates of filesystem/level.

SEE ALSO

restor(1), dump(5), dumpdir(1), sddate (1M)

DIAGNOSTICS

If the dump requires more than one tape, it will ask you to change tapes. Reply with a new-line when this has been done.

BUGS

Sizes are based on 1600 BPI blocked tape. The raw magtape device has to be used to approach these densities. Read errors on the filesystem are ignored. Write errors on the magtape are usually fatal.

NAME

dumpdir - print the names of files on a dump tape

SYNOPSIS

dumpdir [f filename]

DESCRIPTION

Dumpdir is used to read magtapes dumped with the dump command and list the names and inode numbers of all the files and directories on the tape.

The f option causes filename as the name of the tape instead of the default.

FILES

default tape unit varies with installation
rst*

SEE ALSO

dump(1), restor(1)

DIAGNOSTICS

If the dump extends over more than one tape, it may ask you to change tapes. Reply with a new-line when the next tape has been mounted.

BUGS

There is redundant information on the tape that could be used in case of tape reading problems. Unfortunately, dumpdir doesn't use it.

NAME

factor, primes - factor a number, generate large primes

SYNTAX

factor [number]

primes

DESCRIPTION

When factor is invoked without an argument, it waits for a number to be typed in. If you type in a positive number less than 2^{56} (about 7.2e16) it will factor the number and print its prime factors; each one is printed the proper number of times. Then it waits for another number. It exits if it encounters a zero or any non-numeric character.

If factor is invoked with an argument, it factors the number as above and then exits.

Maximum time to factor is proportional to sqrt(n) and occurs when n is prime or the square of a prime. It takes 1 minute to factor a prime near 10^{14} on a PDP11.

When primes is invoked, it waits for a number to be typed in. If you type in a positive number less than 2^{56} it will print all primes greater than or equal to this number.

DIAGNOSTICS

`Ouch.' for input out of range or for garbage input.

NAME

file - determine file type

SYNTAX

```
file filename ...

file -f fileofnames
```

DESCRIPTION

File performs a series of tests on each argument in an attempt to classify it. If an argument appears to be ascii, file examines the first 512 bytes and tries to guess its language.

If the first argument is a -f flag, file will take the list of filenames from the file.

For a.out files, the relationship between flags to cc and the file classification is:

cc flag	classification
i	separate
n	pure
s	not "not stripped"
Z	23fixed

NOTES

It often makes mistakes. In particular it often suggests that command files are C programs. Also, programs that begin with comments are described as English text.

NAME

find - find files

SYNTAX

find pathname-list expression

DESCRIPTION

Find recursively descends the directory hierarchy for each pathname in the pathname-list (i.e., one or more pathnames) seeking files that match a boolean expression written in the primaries given below. In the descriptions, the argument n is used as a decimal integer where +n means more than n, -n means less than n and n means exactly n.

-name filename
: True if the filename argument matches the current file name. Normal Shell argument syntax may be used if escaped (watch out for `[', `?' and `*').

-perm onum
: True if the file permission flags exactly match the octal number onum (see chmod(1)). If onum is prefixed by a minus sign, more flag bits (017777, see stat(2)) become significant and the flags are compared: (flags&onum)==onum.

-type c
: True if the type of the file is c, where c is b, c, d or f for block special file, character special file, directory or plain file.

-links n
: True if the file has n links.

-user uname
: True if the file belongs to the user uname (login name or numeric user ID).

-group gname
: True if the file belongs to group gname (group name or numeric group ID).

-size n
: True if the file is n blocks long (512 bytes per block).

-inum n
: True if the file has inode number n.

-atime n
: True if the file has been accessed in n days.

-mtime n
: True if the file has been modified in n days.

-exec command
: True if the executed command returns a zero value as exit status. The end of the command must be

punctuated by an escaped semicolon. A command argument `{}' is replaced by the current pathname.

-ok command
Like **-exec** except that the generated command is written on the standard output, then the standard input is read and the command executed only upon response **y**.

-print Always true; causes the current pathname to be printed.

-newer file
True if the current file has been modified more recently than the argument file.

The primaries may be combined using the following operators (in order of decreasing precedence):

1) A parenthesized group of primaries and operators (parentheses are special to the Shell and must be escaped).

2) The negation of a primary (`!' is the unary not operator).

3) Concatenation of primaries (the and operation is implied by the juxtaposition of two primaries).

4) Alternation of primaries (`-o' is the or operator).

EXAMPLE

To remove all files named `a.out' or `*.o' that have not been accessed for a week:

```
find / \( -name a.out -o -name '*.o' \) -atime +7 -exec rm
{} \;
```

FILES

/etc/passwd
/etc/group

SEE ALSO

sh(1), test(1), filsys(5)

NOTES

The syntax is painful.

NAME

fsck - file system consistency check and interactive repair

SYNOPSIS

fsck [option] ... [filesystem] ...

DESCRIPTION

Fsck audits and interactively repairs inconsistent conditions for the named filesystems. Fsck ignores the `file system clean' flag in the super block; upon completion fsck sets `file system clean' (if it was not already set).

If a file system is consistent then the number of files, number of blocks used, and number of blocks free are reported. If the file system is inconsistent the operator is prompted for concurrence before each correction is attempted. Most corrections lose data; all losses are reported. The default action for each correction is to wait for the operator to respond `yes' or `no'. Without write permission fsck defaults to -n action.

These options are recognized:

-y Assume a yes response to all questions.

-n Assume a no response to all questions.

-sX Ignore the actual free list and (unconditionally) construct a new one by rewriting the super-block of the file system. The file system should be unmounted while this is done, or extreme care should be taken that the system is quiescent and that it is rebooted immediately afterwards. This precaution is necessary so that the old, bad, in-core copy of the superblock will not continue to be used, or written on the file system.

The free list is created with optimal interleaving according to the specification X:

-s3 optimal for RP03
-s4 optimal for RP04, RP05, RP06
-sc:s space free blocks s blocks apart in cylinders of c blocks each.

If X is not given, the values used when the filesystem was created are used. If these values were not specified, then c=400, s=9 is assumed.

-SX Conditionally reconstruct the free list. This option is like -sX except that the free list is rebuilt only if there were no discrepancies discovered in the file system. It is useful for forcing free list

reorganization on uncontaminated file systems. **-S** forces **-n**.

-t If fsck cannot obtain enough memory to keep its tables, it uses a scratch file. If the **-t** option is specified, the file named in the next argument is used as the scratch file. Without the **-t** option, fsck prompts if it needs a scratch file. The file should not be on the file system being checked, and if it is not a special file or did not already exist, it is removed when fsck completes.

If no filesystems are given to fsck then a default list of file systems is read from the file **/etc/checklist**.

Inconsistencies checked are as follows:

1. Blocks claimed by more than one inode or the free list.

2. Blocks claimed by an inode or the free list outside the range of the file system.

3. Incorrect link counts.

4. Size checks:
   Incorrect number of blocks in file.
   Directory size not a multiple of 16 bytes.

5. Bad inode format.

6. Blocks not accounted for anywhere.

7. Directory checks:
   File pointing to unallocated inode.
   Inode number out of range.

8. Super Block checks:
   More than 65536 inodes.
   More blocks for inodes than there are in the file system.

9. Bad free block list format.

10. Total free block and/or free inode count incorrect.

Orphaned files and directories (allocated but unreferenced) are, with the operator's concurrence, reconnected by placing them in the ``lost+found'' directory. The name assigned is the inode number. The only restriction is that the directory ``lost+found'' must preexist in the root of the filesystem being checked and must have empty slots in which entries can be made. This is accomplished by making ``lost+found'',

copying a number of files to the directory, and then removing them (before fsck is executed).

Checking the raw device is almost always faster.

FILES

/etc/checklist default list of file systems to check.
lost+found home for orphans

SEE ALSO

dcheck(1), icheck(1), filsys(5), crash(8), mount(1M)
/etc/rc the system startup script which uses fsck heavily.

BUGS

Inode numbers for . and .. in each directory should be checked for validity.
The -b option of icheck(1) should be available.

NAME

graph - draw a graph

SYNOPSIS

graph [option] ...

DESCRIPTION

Graph with no options takes pairs of numbers from the standard input as abscissas and ordinates of a graph. Successive points are connected by straight lines. The graph is encoded on the standard output for display by the plot(1) filters.

If the coordinates of a point are followed by a nonnumeric string, that string is printed as a label beginning on the point. Labels may be surrounded with quotes "...", in which case they may be empty or contain blanks and numbers; labels never contain newlines.

The following options are recognized, each as a separate argument.

-a Supply abscissas automatically (they are missing from the input); spacing is given by the next argument (default 1). A second optional argument is the starting point for automatic abscissas (default 0 or lower limit given by **-x**).

-b Break (disconnect) the graph after each label in the input.

-c Character string given by next argument is default label for each point.

-g Next argument is grid style, 0 no grid, 1 frame with ticks, 2 full grid (default).

-l Next argument is label for graph.

-m Next argument is mode (style) of connecting lines: 0 disconnected, 1 connected (default). Some devices give distinguishable line styles for other small integers.

-s Save screen, don't erase before plotting.

-x [l]
If **l** is present, x axis is logarithmic. Next 1 (or 2) arguments are lower (and upper) x limits. Third argument, if present, is grid spacing on x axis. Normally these quantities are determined automatically.

-y [l]

Similarly for y.

-h Next argument is fraction of space for height.

-w Similarly for width.

-r Next argument is fraction of space to move right before plotting.

-u Similarly to move up before plotting.

-t Transpose horizontal and vertical axes. (Option -x now applies to the vertical axis.)

A legend indicating grid range is produced with a grid unless the -s option is present.

If a specified lower limit exceeds the upper limit, the axis is reversed.

SEE ALSO

spline(1), plot(1)

BUGS

Graph stores all points internally and drops those for which there isn't room.
Segments that run out of bounds are dropped, not windowed.
Logarithmic axes may not be reversed.

NAME

icheck - file system storage consistency check

SYNOPSIS

icheck [-s] [-b numbers] [filesystem]

DESCRIPTION

Icheck examines a file system, builds a bit map of used blocks, and compares this bit map against the free list maintained on the file system. If the file system is not specified, a set of default file systems is checked. The normal output of icheck includes a report of

The total number of files and the numbers of regular, directory, block special and character special files.

The total number of blocks in use and the numbers of single-, double-, and triple-indirect blocks and directory blocks.

The number of free blocks.

The number of blocks missing; i.e. not in any file nor in the free list.

The **-s** option causes icheck to ignore the actual free list and reconstruct a new one by rewriting the super-block of the file system. The file system should be dismounted while this is done; if this is not possible (for example if the root file system has to be salvaged) care should be taken that the system is quiescent and that it is rebooted immediately afterwards so that the old, bad in-core copy of the super-block will not continue to be used. Notice also that the words in the super-block which indicate the size of the free list and of the i-list are believed. If the super-block has been curdled these words will have to be patched. The **-s** option causes the normal output reports to be suppressed.

Following the **-b** option is a list of block numbers; whenever any of the named blocks turns up in a file, a diagnostic is produced.

Icheck is faster if the raw version of the special file is used, since it reads the i-list many blocks at a time.

FILES

Default file systems vary with installation.

SEE ALSO

dcheck(1), ncheck(1), filsys(5), clri(1)

DIAGNOSTICS

For duplicate blocks and bad blocks (which lie outside the file system) icheck announces the difficulty, the i-number, and the kind of block involved. If a read error is encountered, the block number of the bad block is printed and icheck considers it to contain 0. `Bad freeblock' means that a block number outside the available space was encountered in the free list. `n dups in free' means that n blocks were found in the free list which duplicate blocks either in some file or in the earlier part of the free list.

BUGS

Since icheck is inherently two-pass in nature, extraneous diagnostics may be produced if applied to active file systems.

It believes even preposterous super-blocks and consequently can get core images.

NAME

iostat - report I/O statistics

SYNOPSIS

iostat [option] ... [interval [count]]

DESCRIPTION

Iostat delves into the system and reports certain statistics kept about input-output activity. Information is kept about up to three different disks (RF, RK, RP) and about typewriters. For each disk, IO completions and number of words transferred are counted; for typewriters collectively, the number of input and output characters are counted. Also, each sixtieth of a second, the state of each disk is examined and a tally is made if the disk is active. The tally goes into one of four categories, depending on whether the system is executing in user mode, in `nice' (background) user mode, in system mode, or idle. From all these numbers and from the known transfer rates of the devices it is possible to determine information such as the degree of IO overlap and average seek times for each device.

The optional interval argument causes iostat to report once each interval seconds. The first report is for all time since a reboot and each subsequent report is for the last interval only.

The optional count argument restricts the number of reports.

With no option argument iostat reports for each disk the number of transfers per minute, the milliseconds per average seek, and the milliseconds per data transfer exclusive of seek time. It also gives the percentage of time the system has spend in each of the four categories mentioned above.

The following options are available:

-t Report the number of characters of terminal IO per second as well.

-i Report the percentage of time spend in each of the four categories mentioned above, the percentage of time each disk was active (seeking or transferring), the percentage of time any disk was active, and the percentage of time spent in `IO wait:' idle, but with a disk active.

-s Report the raw timing information: 32 numbers indicating the percentage of time spent in each of the possible configurations of 4 system states and 8 IO states (3 disks each active or not).

-b Report on the usage of IO buffers.

FILES

/dev/mem, /xenix

BUGS

This program is very configuration dependent and will have to be modified by every installation.

NAME

join - relational database operator

SYNTAX

join [options] file1 file2

DESCRIPTION

Join forms, on the standard output, a join of the two relations specified by the lines of file1 and file2. If file1 is `-', the standard input is used.

File1 and file2 must be sorted in increasing ASCII collating sequence on the fields on which they are to be joined, normally the first in each line.

There is one line in the output for each pair of lines in file1 and file2 that have identical join fields. The output line normally consists of the common field, then the rest of the line from file1, then the rest of the line from file2.

Fields are normally separated by blank, tab or newline. In this case, multiple separators count as one, and leading separators are discarded.

These options are recognized:

-an In addition to the normal output, produce a line for each unpairable line in file n, where n is 1 or 2.

-e s Replace empty output fields by string s.

-jn m
Join on the mth field of file n. If n is missing, use the mth field in each file.

-o list
Each output line comprises the fields specifed in list, each element of which has the form n.m, where n is a file number and m is a field number.

-tc Use character c as a separator (tab character). Every appearance of c in a line is significant.

SEE ALSO

sort(1), comm(1), awk(1)

NOTES

With default field separation, the collating sequence is that of sort -b; with -t, the sequence is that of a plain sort.

The conventions of join, sort, comm, uniq, look and awk(1) are wildly incongruous.

NAME

ln - make a link

SYNTAX

ln name1 [name2]

DESCRIPTION

A link is a directory entry referring to a file; the same file (together with its size, all its protection information, etc.) may have several links to it. There is no way to distinguish a link to a file from its original directory entry; any changes in the file are effective independently of the name by which the file is known.

Ln creates a link to an existing file name1. If name2 is given, the link has that name; otherwise it is placed in the current directory and its name is the last component of name1.

It is forbidden to link to a directory or to link across file systems.

SEE ALSO

rm(1)

NAME

login - sign on

SYNTAX

login [username]

DESCRIPTION

The login command is used when a user initially signs on, or it may be used at any time to change from one user to another. The latter case is the one summarized above and described here. See `How to Get Started' for how to dial up initially.

If login is invoked without an argument, it asks for a user name, and, if appropriate, a password. Echoing is turned off (if possible) during the typing of the password, so it will not appear on the written record of the session.

After a successful login, accounting files are updated and the user is informed of the existence of .mail and message-of-the-day files. Login initializes the user and group IDs and the working directory, then executes a command interpreter (usually sh(1)) according to specifications found in a password file. Argument 0 of the command interpreter is `-sh.

Login is recognized by sh(1) and executed directly (without forking).

FILES

/etc/utmp	accounting
/usr/adm/wtmp	accounting
/usr/mail/*	mail
/etc/motd	message-of-the-day
/etc/passwd	password file

SEE ALSO

init(8), newgrp(1), getty(8), mail(1), passwd(1), passwd(5)

DIAGNOSTICS

`Login incorrect,' if the name or the password is bad.
`No Shell', `cannot open password file', `no directory': consult a programming counselor.

NAME

look - find lines in a sorted list

SYNTAX

look [**-df**] string [file]

DESCRIPTION

Look consults a sorted file and prints all lines that begin with string. It uses binary search.

The options **d** and **f** affect comparisons as in sort(1):

d `Dictionary' order: only letters, digits, tabs and blanks participate in comparisons.

f Fold. Upper case letters compare equal to lower case.

If no file is specified, /usr/dict/words is assumed with collating sequence **-df**.

FILES

/usr/dict/words

SEE ALSO

sort(1), grep(1)

NAME

lorder - find ordering relation for an object library

SYNTAX

lorder file ...

DESCRIPTION

The input is one or more object or library archive (see ar(1)) files. The standard output is a list of pairs of object file names, meaning that the first file of the pair refers to external identifiers defined in the second. The output may be processed by tsort(1) to find an ordering of a library suitable for one-pass access by ld(1).

This brash one-liner intends to build a new library from existing `.o' files.

```
ar cr library `lorder *.o | tsort`
```

FILES

*symref, *symdef
nm(1), sed(1), sort(1), join(1)

SEE ALSO

tsort(1), ld(1), ar(1)

NOTES

The names of object files, in and out of libraries, must end with `.o'; nonsense results otherwise.

NAME

mkconf - generate configuration tables

SYNOPSIS

/sys/conf/mkconf

DESCRIPTION

Mkconf examines a machine configuration table on its standard input. Its output is three files; l.s, c.c and mch0.s. L.s is an assembler program that represents the interrupt vectors located in low memory addresses and the device register addresses. C.c contains initialized block and character device switch tables, a switch table for line protocols and declarations of various configuration dependent and parameterized variables. Mch0.s contains conditional assembly switches which define the tape controller to be used for system crash dumps.

Input to mkconf is a sequence of lines. The following describe devices on the machine:

```
lp    (LP11)
rf    (RS11)
tc    (TU56)
rk    (RK03/RK05)
tm    (TU10/TE10)
rp    (RP03)
hp    (RP04/5/6/RM02/3)
ht    (TU16/TE16)
ts    (TS11)
rx    (RX01/2)
hk    (RK06/7)
rl    (RL01/2)
dc*   (DC11)
kl*   (KL11/DL11-ABC)
dl*   (DL11-E)
dn*   (DN11)
dh*   (DH11)
dhdm*      (DM11-BB)
du*   (DU11)
dz*   (DZ11)
```

The devices marked with * may be preceded by a number telling how many are to be included. The console typewriter is automatically included; don't count it as part of the KL or DL specification. Count DN's in units of 4 (1 system unit).

The following lines are also accepted.

root dev minor
The specified block device (e.g. hp) is used for the root. minor is a decimal number giving the minor

device. This line must appear exactly once.

swap dev minor
The specified block device is used for swapping. If not given the root is used.

pipe dev minor
The specified block device is used to store pipes. If not given the root is used.

swplo number

nswap number
Sets the origin (block number) and size of the area used for swapping. By default, the not very useful numbers 4000 and 872.

time zone dst
Change the default timezone to be zone. Zone may be the name of any timezone in the continental U.S. or the number of minutes westward of Greenwich. Dst should be 1 if the daylight savings time conversion should be done.

hertz num
The line clock frequency is set to num Hertz. The default value is taken from the parameter DHZ in param.h.

nbufs num
The number of system buffers is set to num. The default value is taken from the parameter DNBUF in param.h.

pack Include the packet driver. By default it is left out.

mpx Include the multiplexor driver. By default it is left out.

FILES

l.s, c.c, mch0.s output files

SEE ALSO

configure(1m)
Device driver descriptions in section 4.
`Setting up XENIX', in Volume 2B.

BUGS

Because of floating vectors that may have been missed, it is mandatory to check the l.s file to make sure it corresponds with reality.

NAME

mkfs - construct a file system

SYNOPSIS

/etc/mkfs special proto [m n]

DESCRIPTION

Mkfs constructs a file system by writing on the special file special according to the directions found in the prototype file proto. The prototype file contains tokens separated by spaces or new lines. The first token is the name of a file to be copied onto block zero as the bootstrap program, see bproc(8). The second token is a number specifying the size of the created file system. Typically it will be the number of blocks on the device, perhaps diminished by space for swapping. The next token is the number of i-nodes in the i-list. The next set of tokens comprise the specification for the root file. File specifications consist of tokens giving the mode, the user-id, the group id, and the initial contents of the file. The syntax of the contents field depends on the mode.

The mode token for a file is a 6 character string. The first character specifies the type of the file. (The characters **-bcd** specify regular, block special, character special and directory files respectively.) The second character of the type is either **u** or **-** to specify set-user-id mode or not. The third is **g** or **-** for the set-group-id mode. The rest of the mode is a three digit octal number giving the owner, group, and other read, write, execute permissions, see chmod(1).

Two decimal number tokens come after the mode; they specify the user and group ID's of the owner of the file.

If the file is a regular file, the next token is a pathname whence the contents and size are copied.

If the file is a block or character special file, two decimal number tokens follow which give the major and minor device numbers.

If the file is a directory, mkfs makes the entries **.** and **..** and then reads a list of names and (recursively) file specifications for the entries in the directory. The scan is terminated with the token **$**.

If the prototype file cannot be opened and its name consists of a string of digits, mkfs builds a file system with a single empty directory on it. The size of the file system is the value of proto interpreted as a decimal number. The number of i-nodes is calculated as a function of the

filsystem size. The boot program is left uninitialized.

A sample prototype specification follows:

```
/usr/mdec/uboot
4872 55
d--777 3 1
usr  d--777 3 1
     sh   ---755 3 1 /bin/sh
     ken  d--755 6 1
          $
     b0   b--644 3 1 0 0
     c0   c--644 3 1 0 0
     $
$
```

SEE ALSO

filsys(5), dir(5), bproc(8)

BUGS

There should be some way to specify links.

NAME

mknod - build special file

SYNOPSIS

/etc/mknod name [**c**] [**b**] major minor

DESCRIPTION

Mknod makes a special file. The first argument is the name of the entry. The second is **b** if the special file is block-type (disks, tape) or **c** if it is character-type (other devices). The last two arguments are numbers specifying the major device type and the minor device (e.g. unit, drive, or line number).

The assignment of major device numbers is specific to each system. They have to be dug out of the system source file conf.c.

SEE ALSO

mknod(2)

NAME

mount, umount - mount and dismount file system

SYNOPSIS

/etc/mount [special name [-r]]

/etc/umount special

DESCRIPTION

Mount announces to the system that a removable file system is present on the device special. The file name must exist already; it must be a directory (unless the root of the mounted file system is not a directory). It becomes the name of the newly mounted root. The optional last argument indicates that the file system is to be mounted read-only.

Umount announces to the system that the removable file system previously mounted on device special is to be removed. First, any pending I/O for the file system is completed, and the file system is flagged clean. Mount will refuse to mount a file system which is not flagged clean; this can happen if a system crash prevented the use of umount or haltsys(8). In such a case, use fsck(1M) to clean the file system, then try mount again.

These commands maintain a table of mounted devices. If invoked without an argument, mount prints the table.

Physically write-protected and magnetic tape file systems must be mounted read-only or errors will occur when access times are updated, whether or not any explicit write is attempted.

FILES

/etc/mtab: mount table

SEE ALSO

mount(2), mtab(5)

DIAGNOSTICS

Exit code 0 is returned for a successful mount, 1 for a failure, 2 for attempting to mount an unclean structure.

BUGS

Mounting file systems full of garbage will crash the system. Mounting a root directory on a non-directory makes some apparently good pathnames invalid.

NAME

ncheck - generate names from i-numbers

SYNOPSIS

ncheck [-i numbers] [-a] [-s] [filesystem]

DESCRIPTION

Ncheck with no argument generates a pathname vs. i-number list of all files on a set of default file systems. Names of directory files are followed by `/.'. The **-i** option reduces the report to only those files whose i-numbers follow. The **-a** option allows printing of the names `.' and `..', which are ordinarily suppressed. The **-s** option reduces the report to special files and files with set-user-ID mode; it is intended to discover concealed violations of security policy.

A file system may be specified.

The report is in no useful order, and probably should be sorted.

SEE ALSO

dcheck(1), icheck(1), sort(1)

DIAGNOSTICS

When the filesystem structure is improper, `??' denotes the `parent' of a parentless file and a pathname beginning with `...' denotes a loop.

NAME

nm - print name list

SYNTAX

nm [-gnoprucx] [file ...]

DESCRIPTION

Nm prints the name list (symbol table) of each object file in the argument list. If an argument is an archive, a listing for each object file in the archive will be produced. If no file is given, the symbols in `a.out' are listed.

Each symbol name is preceded by its value (blanks if undefined) and one of the letters **U** (undefined), **A** (absolute), **T** (text segment symbol), **D** (data segment symbol), **B** (bss segment symbol), or **C** (common symbol). If the symbol is local (non-external) the type letter is in lower case. The output is sorted alphabetically.

Options are:

-g Print only global (external) symbols.

-n Sort numerically rather than alphabetically.

-o Prepend file or archive element name to each output line rather than only once.

-p Don't sort; print in symbol-table order.

-r Sort in reverse order.

-u Print only undefined symbols.

-c Print only C program symbols (symbols which begin with `_') as they appeared in the C program.

-x Symbol values are printed in hexadecimal rather than octal.

FILES

a.out Default input file.

SEE ALSO

ar(1), ar(5), a.out(5)

NAME

plot - graphics filters

SYNOPSIS

plot [-Tterminal [raster]]

DESCRIPTION

These commands read plotting instructions (see plot(5)) from the standard input, and in general produce plotting instructions suitable for a particular terminal on the standard output.

If no terminal type is specified, the environment parameter $TERM (see environ(5)) is used. Known terminals are:

4014 Tektronix 4014 storage scope.

450 DASI Hyterm 450 terminal (Diablo mechanism).

300 DASI 300 or GSI terminal (Diablo mechanism).

300S DASI 300S terminal (Diablo mechanism).

ver Versatec D1200A printer-plotter. This version of plot places a scan-converted image in `/usr/tmp/raster' and sends the result directly to the plotter device rather than to the standard output. The optional argument causes a previously scan-converted file raster to be sent to the plotter.

FILES

/usr/bin/tek
/usr/bin/t450
/usr/bin/t300
/usr/bin/t300s
/usr/bin/vplot
/usr/tmp/raster

SEE ALSO

plot(3), plot(5)

BUGS

There is no lockout protection for /usr/tmp/raster.

NAME

pstat - print system facts

SYNOPSIS

pstat [-aixptuf] [suboptions] [file]

DESCRIPTION

Pstat interprets the contents of certain system tables. If file is given, the tables are sought there, otherwise in /dev/mem. The required namelist is taken from /xenix. Options are

-a Under **-p**, describe all process slots rather than just active ones.

-i Print the inode table with the these headings:

LOC The core location of this table entry.

FLAGS Miscellaneous state variables encoded thus:

- L locked
- U update time filsys(5)) must be corrected
- A access time must be corrected
- M file system is mounted here
- W wanted by another process (L flag is on)
- T contains a text file
- C changed time must be corrected

CNT Number of open file table entries for this inode.

DEV Major and minor device number of file system in which this inode resides.

INO I-number within the device.

MODE Mode bits, see chmod(2).

NLK Number of links to this inode.

UID User ID of owner.

SIZ/DEV Number of bytes in an ordinary file, or major and minor device of special file.

-x Print the text table with these headings:

LOC The core location of this table entry.

FLAGS Miscellaneous state variables encoded thus:

- T ptrace(2) in effect
- W text not yet written on swap device
- L loading in progress
- K locked
- w wanted (L flag is on)

DADDR Disk address in swap, measured in multiples of 512 bytes.

CADDR Core address, measured in multiples of 64 bytes.

SIZE Size of text segment, measured in multiples of 64 bytes.

IPTR Core location of corresponding inode.

INUM Inode number of corresponding inode (executable file).

CNT Number of processes using this text segment.

CCNT Number of processes in core using this text segment.

-p Print process table for active processes with these headings:

LOC The core location of this table entry.
S Run state encoded thus:
0 no process
1 waiting for some event
3 runnable
4 being created
5 being terminated
6 stopped under trace
F Miscellaneous state variables, or-ed together:
01 loaded
02 the scheduler process
04 locked
010 swapped out
020 traced
040 used in tracing
0100 locked in by lock(2).
PRI Scheduling priority, see nice(2).
SIGNAL
Signals received (signals 1-16 coded in bits 0-15),
UID Real user ID.
TIM Time resident in seconds; times over 127 coded as 127.
CPU Weighted integral of CPU time, for scheduler.
NI Nice level, see nice(2).
PGRP Process number of root of process group (the opener of the controlling terminal).
PID The process ID number.
PPID The process ID of parent process.
ADDR If in core, the physical address of the `u-area' of the process measured in multiples of 64 bytes. If swapped out, the position in the swap area measured in multiples of 512 bytes.
SIZE Size of process image in multiples of 64 bytes.
WCHAN Wait channel number of a waiting process.
LINK Link pointer in list of runnable processes.
TEXTP If text is pure, pointer to location of text table entry.
CLKT Countdown for alarm(2) measured in seconds.

NAME

pstat - print system facts

SYNOPSIS

pstat [-aixptuf] [suboptions] [file]

DESCRIPTION

Pstat interprets the contents of certain system tables. If file is given, the tables are sought there, otherwise in /dev/mem. The required namelist is taken from /xenix. Options are

-a Under **-p**, describe all process slots rather than just active ones.

-i Print the inode table with the these headings:

LOC The core location of this table entry.
FLAGS Miscellaneous state variables encoded thus:
- L locked
- U update time filsys(5)) must be corrected
- A access time must be corrected
- M file system is mounted here
- W wanted by another process (L flag is on)
- T contains a text file
- C changed time must be corrected

CNT Number of open file table entries for this inode.
DEV Major and minor device number of file system in which this inode resides.
INO I-number within the device.
MODE Mode bits, see chmod(2).
NLK Number of links to this inode.
UID User ID of owner.
SIZ/DEV
Number of bytes in an ordinary file, or major and minor device of special file.

-x Print the text table with these headings:

LOC The core location of this table entry.
FLAGS Miscellaneous state variables encoded thus:
- T ptrace(2) in effect
- W text not yet written on swap device
- L loading in progress
- K locked
- w wanted (L flag is on)

DADDR Disk address in swap, measured in multiples of 512 bytes.

CADDR Core address, measured in multiples of 64 bytes.

SIZE Size of text segment, measured in multiples of 64 bytes.

IPTR Core location of corresponding inode.

INUM Inode number of corresponding inode (executable file).

CNT Number of processes using this text segment.

CCNT Number of processes in core using this text segment.

-p Print process table for active processes with these headings:

LOC The core location of this table entry.
S Run state encoded thus:
0 no process
1 waiting for some event
3 runnable
4 being created
5 being terminated
6 stopped under trace
F Miscellaneous state variables, or-ed together:
01 loaded
02 the scheduler process
04 locked
010 swapped out
020 traced
040 used in tracing
0100 locked in by lock(2).
PRI Scheduling priority, see nice(2).
SIGNAL
Signals received (signals 1-16 coded in bits 0-15),
UID Real user ID.
TIM Time resident in seconds; times over 127 coded as 127.
CPU Weighted integral of CPU time, for scheduler.
NI Nice level, see nice(2).
PGRP Process number of root of process group (the opener of the controlling terminal).
PID The process ID number.
PPID The process ID of parent process.
ADDR If in core, the physical address of the `u-area' of the process measured in multiples of 64 bytes. If swapped out, the position in the swap area measured in multiples of 512 bytes.
SIZE Size of process image in multiples of 64 bytes.
WCHAN Wait channel number of a waiting process.
LINK Link pointer in list of runnable processes.
TEXTP If text is pure, pointer to location of text table entry.
CLKT Countdown for alarm(2) measured in seconds.

-t Print table for terminals (only DH11 and DL11 handled) with these headings:

RAW Number of characters in raw input queue.
CAN Number of characters in canonicalized input queue.
OUT Number of characters in putput queue.
MODE See tty(4).
ADDR Physical device address.
DEL Number of delimiters (newlines) in canonicalized input queue.
COL Calculated column position of terminal.
STATE Miscellaneous state variables encoded thus:
- W waiting for open to complete
- O open
- S has special (output) start routine
- C carrier is on
- B busy doing output
- A process is awaiting output
- X open for exclusive use
- H hangup on close

PGRP Process group for which this is controlling terminal.

-u print information about a user process; the next argument is its address as given by ps(1). The process must be in main memory, or the file used can be a core image and the address 0.

-f Print the open file table with these headings:

LOC The core location of this table entry.
FLG Miscellaneous state variables encoded thus:
- R open for reading
- W open for writing
- P pipe

CNT Number of processes that know this open file.
INO The location of the inode table entry for this file.
OFFS The file offset, see lseek(2).

FILES

/xenix namelist
/dev/mem default source of tables

SEE ALSO

ps(1), stat(2), filsys(5)
K. Thompson, UNIX Implementation

NAME

quot - summarize file system ownership

SYNOPSIS

quot [option] ... [filesystem]

DESCRIPTION

Quot prints the number of blocks in the named filesystem currently owned by each user. If no filesystem is named, a default name is assumed. The following options are available:

-n Cause the pipeline **ncheck filesystem | sort +0n | quot -n filesystem** to produce a list of all files and their owners.

-c Print three columns giving file size in blocks, number of files of that size, and cumulative total of blocks in that size or smaller file.

-f Print count of number of files as well as space owned by each user.

FILES

Default file system varies with system.
/etc/passwd to get user names

SEE ALSO

ls(1), du(1)

BUGS

Holes in files are counted as if they actually occupied space.

NAME

restor - incremental file system restore

SYNOPSIS

restor key [argument ...]

DESCRIPTION

Restor is used to read magtapes dumped with the dump command. The key specifies what is to be done. Key is one of the characters **rRxt** optionally combined with **f**.

f Use the first argument as the name of the tape instead of the default.

r or R
The tape is read and loaded into the file system specified in argument. This should not be done lightly (see below). If the key is **R** restor asks which tape of a multi volume set to start on. This allows restor to be interrupted and then restarted (an icheck -s must be done before restart).

x Each file on the tape named by an argument is extracted. The file name has all `mount' prefixes removed; for example, /usr/bin/lpr is named /bin/lpr on the tape. The file extracted is placed in a file with a numeric name supplied by restor (actually the inode number). In order to keep the amount of tape read to a minimum, the following procedure is recommended:

Mount volume 1 of the set of dump tapes.

Type the restor command.

Restor will announce whether or not it found the files, give the number it will name the file, and rewind the tape.

It then asks you to `mount the desired tape volume'. Type the number of the volume you choose. On a multi volume dump the recommended procedure is to mount the last through the first volume in that order. Restor checks to see if any of the files requested are on the mounted tape (or a later tape, thus the reverse order) and doesn't read through the tape if no files are. If you are working with a single volume dump or the number of files being restored is large, respond to the query with `1' and restor will read the tapes in sequential order.

If you have a hierarchy to restore you can use dumpdir(1) to produce the list of names and a shell script

to move the resulting files to their homes.

t Print the date the tape was written and the date the filesystem was dumped from.

The **r** option should only be used to restore a complete dump tape onto a clear file system or to restore an incremental dump tape onto this. Thus

```
/etc/mkfs /dev/rp0 40600
restor r /dev/rp0
```

is a typical sequence to restore a complete dump. Another restor can be done to get an incremental dump in on top of this.

A dump followed by a mkfs and a restor is used to change the size of a file system.

FILES

default tape unit varies with installation
rst*

SEE ALSO

dump(1), mkfs(1), dumpdir(1)

DIAGNOSTICS

There are various diagnostics involved with reading the tape and writing the disk. There are also diagnostics if the i-list or the free list of the file system is not large enough to hold the dump.

If the dump extends over more than one tape, it may ask you to change tapes. Reply with a new-line when the next tape has been mounted.

BUGS

There is redundant information on the tape that could be used in case of tape reading problems. Unfortunately, restor doesn't use it.

NAME

sa, accton - system accounting

SYNOPSIS

sa [-abcijlnrstuv] [file]

/etc/accton [file]

DESCRIPTION

With an argument naming an existing file, accton causes system accounting information for every process executed to be placed at the end of the file. If no argument is given, accounting is turned off.

Sa reports on, cleans up, and generally maintains accounting files.

Sa is able to condense the information in /usr/adm/acct into a summary file /usr/adm/savacct which contains a count of the number of times each command was called and the time resources consumed. This condensation is desirable because on a large system acct can grow by 100 blocks per day. The summary file is read before the accounting file, so the reports include all available information.

If a file name is given as the last argument, that file will be treated as the accounting file; /usr/adm/acct is the default. There are zillions of options:

a Place all command names containing unprintable characters and those used only once under the name `***other.'

b Sort output by sum of user and system time divided by number of calls. Default sort is by sum of user and system times.

c Besides total user, system, and real time for each command print percentage of total time over all commands.

i Ignore the summary files /usr/adm/savacct and /usr/adm/usracct; do not include their contents in this report.

j Instead of total minutes time for each category, give seconds per call.

l Separate system and user time; normally they are combined.

m Print number of processes and number of CPU minutes for each user.

n Sort by number of calls.

r Reverse order of sort.

s Merge accounting file into summary file /usr/adm/savacct when done.

t For each command report ratio of real time to the sum of user and system times.

u Superseding all other flags, print for each command in the accounting file the user ID and command name.

v If the next character is a digit n, then type the name of each command used n times or fewer. Await a reply from the typewriter; if it begins with `y', add the command to the category `**junk**.' This is used to strip out garbage.

(default)
A table of 4 columns is printed: the number of calls, the total real time, the total combined system and user time, and the name of the command. The first line in the table contains the sum of each column.

FILES

/usr/adm/acct raw accounting
/usr/adm/savacct summary
/usr/adm/usracct per-user summary

SEE ALSO

ac(1), acct(2)

NAME

sddate - print and set dump dates

SYNOPSIS

sddate [name lev date]

DESCRIPTION

If no argument is given, the contents of the dump date file `/etc/ddate' are printed. The dump date file is maintained by dump(1M) and contains the date of the most recent dump for each dump level for each filesystem.

If arguments are given, an entry is replaced or made in `/etc/ddate'. name is the last component of the device pathname. lev is the dump level number (from 0 to 9), and date is a time in the form taken by date(1).

Some sites may wish to backup filesystems by coping them verbatim to dismountable packs. Sddate could be used to make a `level 0' entry in `/etc/ddate', which would then allow incremental mag tape dumps.

For example:

```
sddate rrp3 5 10081520
```

makes an `/etc/ddate' entry showing a level 5 dump of `/dev/rrp3' on October 8, at 3:20 PM.

FILES

/etc/ddate

SEE ALSO

dump(1M), date(1)

DIAGNOSTICS

`bad conversion' if the date set is syntactically incorrect.

NAME

spline - interpolate smooth curve

SYNOPSIS

spline [option] ...

DESCRIPTION

Spline takes pairs of numbers from the standard input as abcissas and ordinates of a function. It produces a similar set, which is approximately equally spaced and includes the input set, on the standard output. The cubic spline output (R. W. Hamming, Numerical Methods for Scientists and Engineers, 2nd ed., 349ff) has two continuous derivatives, and sufficiently many points to look smooth when plotted, for example by graph(1).

The following options are recognized, each as a separate argument.

-a Supply abscissas automatically (they are missing from the input); spacing is given by the next argument, or is assumed to be 1 if next argument is not a number.

-k The constant k used in the boundary value computation

(2nd deriv. at end) = k*(2nd deriv. next to end)

is set by the next argument. By default k = 0.

-n Space output points so that approximately n intervals occur between the lower and upper x limits. (Default n = 100.)

-p Make output periodic, i.e. match derivatives at ends. First and last input values should normally agree.

-x Next 1 (or 2) arguments are lower (and upper) x limits. Normally these limits are calculated from the data. Automatic abcissas start at lower limit (default 0).

SEE ALSO

graph(1)

DIAGNOSTICS

When data is not strictly monotone in x, spline reproduces the input without interpolating extra points.

BUGS

A limit of 1000 input points is enforced silently.

NAME

stty - set terminal options

SYNTAX

stty [option ...]

DESCRIPTION

Stty sets certain I/O options on the current output terminal. With no argument, it reports the current settings of the options. The option strings are selected from the following set:

even	allow even parity
-even	disallow even parity
odd	allow odd parity
-odd	disallow odd parity
raw	raw mode input (no erase, kill, interrupt, quit, EOT; parity bit passed back)
-raw	negate raw mode
cooked	same as `-raw'
cbreak	make each character available to read(2) as received; no erase and kill
-cbreak	make characters available to read only when newline is received
-nl	allow carriage return for new-line, and output CR-LF for carriage return or new-line
nl	accept only new-line to end lines
echo	echo back every character typed
-echo	do not echo characters
lcase	map upper case to lower case
-lcase	do not map case
-tabs	replace tabs by spaces when printing
tabs	preserve tabs
ek	reset erase and kill characters back to normal # and @
erase *c*	set erase character to *c*. *C* can be of the form `^*X*' which is interpreted as a `control X'.
kill *c*	set kill character to *c*. `^*X*' works here also.
cr0 cr1 cr2 cr3	select style of delay for carriage return (see ioctl(2))
nl0 nl1 nl2 nl3	select style of delay for linefeed
tab0 tab1 tab2 tab3	select style of delay for tab
ff0 ff1	select style of delay for form feed
bs0 bs1	select style of delay for backspace
tty33	set all modes suitable for the Teletype Corporation Model 33 terminal.
tty37	set all modes suitable for the Teletype Corporation Model 37 terminal.
vt05	set all modes suitable for Digital Equipment Corp.

VT05 terminal

tn300 set all modes suitable for a General Electric TermiNet 300

ti700 set all modes suitable for Texas Instruments 700 series terminal

tek set all modes suitable for Tektronix 4014 terminal

hup hang up dataphone on last close.

-hup do not hang up dataphone on last close.

0 hang up phone line immediately

50 75 110 134 150 200 300 600 1200 1800 2400 4800 9600 exta extb
Set terminal baud rate to the number given, if possible. (These are the speeds supported by the DH-11 interface).

SEE ALSO

ioctl(2), tabs(1)

NAME

su - substitute user id temporarily

SYNTAX

su [userid]

DESCRIPTION

Su demands the password of the specified userid, and if it is given, changes to that userid and invokes the Shell sh(1) without changing the current directory or the user environment (see environ(5)). The new user ID stays in force until the Shell exits.

If no userid is specified, `root' is assumed. To remind the super-user of his responsibilities, the Shell substitutes `#' for its usual prompt.

SEE ALSO

sh(1)

NAME

sum - sum and count blocks in a file

SYNTAX

sum file

DESCRIPTION

Sum calculates and prints a 16-bit checksum for the named file, and also prints the number of blocks in the file. It is typically used to look for bad spots, or to validate a file communicated over some transmission line.

SEE ALSO

wc(1)

DIAGNOSTICS

`Read error' is indistinuishable from end of file on most devices; check the block count.

NAME

sync - update the super block

SYNOPSIS

sync

DESCRIPTION

Sync executes the sync system primitive. If the system is to be stopped, sync must be called to insure file system integrity. See sync(2) for details.

SEE ALSO

sync(2), update(8)

NAME

tabs - set terminal tabs

SYNTAX

tabs [**-n**] [terminal]

DESCRIPTION

Tabs sets the tabs on a variety of terminals. Various of the terminal names given in term(7) are recognized; the default is, however, suitable for most 300 baud terminals. If the **-n** flag is present then the left margin is not indented as is normal.

SEE ALSO

stty(1), term(7)

NAME

tar - tape archiver

SYNTAX

tar [key] [name ...]

DESCRIPTION

Tar saves and restores files on magtape. Its actions are controlled by the key argument. The key is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to the command are file or directory names specifying which files are to be dumped or restored. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

The function portion of the key is specified by one of the following letters:

r The named files are written on the end of the tape. The **c** function implies this.

x The named files are extracted from the tape. If the named file matches a directory whose contents had been written onto the tape, this directory is (recursively) extracted. The owner, modification time, and mode are restored (if possible). If no file argument is given, the entire content of the tape is extracted. Note that if multiple entries specifying the same file are on the tape, the last one overwrites all earlier.

t The names of the specified files are listed each time they occur on the tape. If no file argument is given, all of the names on the tape are listed.

u The named files are added to the tape if either they are not already there or have been modified since last put on the tape.

c Create a new tape; writing begins on the beginning of the tape instead of after the last file. This command implies **r**.

The following characters may be used in addition to the letter which selects the function desired.

0,...,7 This modifier selects the drive on which the tape is mounted. The default is **1**.

v Normally tar does its work silently. The **v** (verbose) option causes it to type the name of each

file it treats preceded by the function letter. With the **t** function, **v** gives more information about the tape entries than just the name.

w causes tar to print the action to be taken followed by file name, then wait for user confirmation. If a word beginning with `y' is given, the action is performed. Any other input means don't do it.

f causes tar to use the next argument as the name of the archive instead of /dev/mt?. If the name of the file is `-', tar writes to standard output or reads from standard input, whichever is appropriate. Thus, tar can be used as the head or tail of a filter chain Tar can also be used to move hierarchies with the command

```
cd fromdir; tar cf - . | (cd todir; tar xf -)
```

b causes tar to use the next argument as the blocking factor for tape records. The default is 1, the maximum is 20. This option should only be used with raw magnetic tape archives (See **f** above). The block size is determined automatically when reading tapes (key letters `x' and `t').

l tells tar to complain if it cannot resolve all of the links to the files dumped. If this is not specified, no error messages are printed.

m tells tar to not restore the modification times. The mod time will be the time of extraction.

s causes tar to use the next argument as the size of a tape volume. The minimum value allowed is 500. This option is useful when the archive is not intended for a magnetic tape device, but for some fixed size device, such as floppy disk (See **f** above).

FILES

/dev/mt?
/tmp/tar*

DIAGNOSTICS

Complaints about bad key characters and tape read/write errors.
Complaints if enough memory is not available to hold the link tables.

EXAMPLES

To backup a disk directory tree to tape using raw I/O and a blocking factor of 20:

```
tar cfb /dev/rmtl 20 directory_name
```

To restore the above files from tape to disk:

```
tar xf /dev/rmtl directory_name
```

SEE ALSO

tp(1), dump(1), restor(1), copy(1), dd(1)

NOTES

There is no way to ask for the n-th occurrence of a file.
Tape errors are handled ungracefully.
The **u** option can be slow.
The **b** option should not be used with archives that are going to be updated. The current magtape driver cannot backspace raw magtape. If the archive is on a disk file the **b** option should not be used at all, as updating an archive stored in this manner can destroy it.
The current limit on file name length is 100 characters.

NAME

tp - manipulate tape archive

SYNTAX

tp [key] [name ...]

DESCRIPTION

Tp saves and restores files on DECtape or magtape. Its actions are controlled by the key argument. The key is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to the command are file or directory names specifying which files are to be dumped, restored, or listed. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

The function portion of the key is specified by one of the following letters:

r The named files are written on the tape. If files with the same names already exist, they are replaced. `Same' is determined by string comparison, so `./abc' can never be the same as `/usr/dmr/abc' even if `/usr/dmr' is the current directory. If no file argument is given, `.' is the default.

u updates the tape. u is like r, but a file is replaced only if its modification date is later than the date stored on the tape; that is to say, if it has changed since it was dumped. u is the default command if none is given.

d deletes the named files from the tape. At least one name argument must be given. This function is not permitted on magtapes.

x extracts the named files from the tape to the file system. The owner and mode are restored. If no file argument is given, the entire contents of the tape are extracted.

t lists the names of the specified files. If no file argument is given, the entire contents of the tape is listed.

The following characters may be used in addition to the letter which selects the function desired.

m Specifies magtape as opposed to DECtape.

0,...,7 This modifier selects the drive on which the tape

is mounted. For DECtape, **x** is default; for magtape `0' is the default.

v Normally tp does its work silently. The **v** (verbose) option causes it to type the name of each file it treats preceded by the function letter. With the **t** function, **v** gives more information about the tape entries than just the name.

c means a fresh dump is being created; the tape directory is cleared before beginning. Usable only with **r** and **u**. This option is assumed with magtape since it is impossible to selectively overwrite magtape.

i Errors reading and writing the tape are noted, but no action is taken. Normally, errors cause a return to the command level.

f Use the first named file, rather than a tape, as the archive. This option is known to work only with **x**.

w causes tp to pause before treating each file, type the indicative letter and the file name (as with **v**) and await the user's response. Response **y** means `yes', so the file is treated. Null response means `no', and the file does not take part in whatever is being done. Response **x** means `exit'; the tp command terminates immediately. In the **x** function, files previously asked about have been extracted already. With **r**, **u**, and **d** no change has been made to the tape.

FILES

/dev/tap?
/dev/mt?

SEE ALSO

ar(1), tar(1)

DIAGNOSTICS

Several; the non-obvious one is `Phase error', which means the file changed after it was selected for dumping but before it was dumped.

NOTES

A single file with several links to it is treated like several files.

Binary-coded control information makes magnetic tapes written by tp difficult to carry to other machines; tar(1)

avoids the problem.

NAME

tty - get terminal name

SYNTAX

tty

DESCRIPTION

Tty prints the pathname of the user's terminal.

DIAGNOSTICS

`not a tty' if the standard input file is not a terminal.

NAME

uucp, uulog - unix to unix copy

SYNOPSIS

uucp [option] ... source-file ... destination-file

uulog [option] ...

DESCRIPTION

Uucp copies files named by the source-file arguments to the destination-file argument. A file name may be a path name on your machine, or may have the form

system-name!pathname

where `system-name' is taken from a list of system names which uucp knows about. Shell metacharacters ?*[] appearing in the pathname part will be expanded on the appropriate system.

Pathnames may be one of

(1) a full pathname;

(2) a pathname preceded by ~user; where user is a userid on the specified system and is replaced by that user's login directory;

(3) anything else is prefixed by the current directory.

If the result is an erroneous pathname for the remote system the copy will fail. If the destination-file is a directory, the last part of the source-file name is used.

Uucp preserves execute permissions across the transmission and gives 0666 read and write permissions (see chmod(2)).

The following options are interpreted by uucp.

-d Make all necessary directories for the file copy.

-c Use the source file when copying out rather than copying the file to the spool directory.

-m Send mail to the requester when the copy is complete.

Uulog maintains a summary log of uucp and uux(1) transactions in the file `/usr/spool/uucp/LOGFILE' by gathering information from partial log files named `/usr/spool/uucp/LOG.*.?'. It removes the partial log files.

The options cause uulog to print logging information:

-ssys
Print information about work involving system sys.

-uuser
Print information about work done for the specified user.

FILES
/usr/spool/uucp - spool directory
/usr/lib/uucp/* - other data and program files

SEE ALSO
uux(1), mail(1)
D. A. Nowitz, Uucp Implementation Description

WARNING
The domain of remotely accessible files can (and for obvious security reasons, usually should) be severely restricted. You will very likely not be able to fetch files by pathname; ask a responsible person on the remote system to send them to you. For the same reasons you will probably not be able to send files to arbitrary pathnames.

BUGS
All files received by uucp will be owned by uucp.
The -m option will only work sending files or receiving a single file. (Receiving multiple files specified by special shell characters ?*[] will not activate the -m option.)

NAME

uux - unix to unix command execution

SYNOPSIS

uux [-] command-string

DESCRIPTION

Uux will gather 0 or more files from various systems, execute a command on a specified system and send standard output to a file on a specified system.

The command-string is made up of one or more arguments that look like a shell command line, except that the command and file names may be prefixed by system-name!. A null system-name is interpreted as the local system.

File names may be one of

(1) a full pathname;

(2) a pathname preceded by ~xxx; where xxx is a userid on the specified system and is replaced by that user's login directory;

(3) anything else is prefixed by the current directory.

The `-' option will cause the standard input to the uux command to be the standard input to the command-string.

For example, the command

```
uux "!diff usg!/usr/dan/f1 pwba!/a4/dan/f1 > !fi.diff"
```

will get the f1 files from the usg and pwba machines, execute a diff command and put the results in f1.diff in the local directory.

Any special shell characters such as <>;| should be quoted either by quoting the entire command-string, or quoting the special characters as individual arguments.

FILES

/usr/uucp/spool - spool directory
/usr/uucp/* - other data and programs

SEE ALSO

uucp(1)
D. A. Nowitz, Uucp implementation description

WARNING

An installation may, and for security reasons generally will, limit the list of commands executable on behalf of an

incoming request from uux. Typically, a restricted site will permit little other than the receipt of mail via uux.

BUGS

Only the first command of a shell pipeline may have a system-name!. All other commands are executed on the system of the first command.
The use of the shell metacharacter * will probably not do what you want it to do.
The shell tokens << and >> are not implemented.
There is no notification of denial of execution on the remote machine.

NAME

wall - write to all users

SYNOPSIS

/etc/wall

DESCRIPTION

Wall reads its standard input until an end-of-file. It then sends this message, preceded by `Broadcast Message ...', to all logged in users.

The sender should be super-user to override any protections the users may have invoked.

FILES

/dev/tty?
/etc/utmp

SEE ALSO

mesg(1), write(1)

DIAGNOSTICS

`Cannot send to ...' when the open on a user's tty file fails.

NAME

xsend, xget, enroll - secret mail

SYNTAX

xsend person
xget
enroll

DESCRIPTION

These commands implement a secure communication channel; it is like mail(1), but no one can read the messages except the intended recipient. The method embodies a public-key cryptosystem using knapsacks.

To receive messages, use enroll; it asks you for a password that you must subsequently quote in order to receive secret mail.

To receive secret mail, use xget. It asks for your password, then gives you the messages.

To send secret mail, use xsend in the same manner as the ordinary mail command. (However, it will accept only one target). A message announcing the receipt of secret mail is also sent by ordinary mail.

FILES

/usr/spool/secretmail/*.key: keys
/usr/spool/secretmail/*.[0-9]: messages

SEE ALSO

mail (1)

NOTES

It should be integrated with ordinary mail. The announcement of secret mail makes traffic analysis possible.

APPENDIX A:

Device Driver Routines

Included here are two device driver routines, for disk and tape respectively, as described in Chapter 3. Keep in mind that these are intended solely as examples of device driver routines; much of the material may not be applicable to a specific machine.

A.1 Prototype disk driver

What follows is a sample disk driver. It does not represent any particular driver but demonstrates features common to most disk drivers. The driver is intended for a controller supporting multiple large drives suitable for a root/swap/usr filesystem. The driver currently assumes that the controller is intelligent enough to perform seeks when required. If this is not the case, an array for current cylinder address would have to be added.

The logical device addressing is set up so that the low-order three bits specify a psuedo-drive number. The psuedo-drives partition the disk into convenient sized chunks that allow multiple filesystems on the disk. For the exact layout of the psuedo-drives see the definition of bd_sizes below. The remainder of the minor number specifies the physical drive number.

```
 */

#include "../h/param.h"          /* SYSTEM PARAMTERS */
#include "../h/systm.h"          /* SYSTEM VARIABLES */
#include "../h/buf.h"            /* BUFFER STRUCTURE */
#include "../h/dir.h"            /* DIRECTORY STRUCTURE */
#include "../h/conf.h"           /* CONFIGURATION TABLE FORMATS */
#include "../h/user.h"           /* USER STRUCTURE */


        /* LAYOUT OF DEVICE REGISTERS */
            /*  DEVICE DEPENDENT  */


struct device {
        int     bdds;           /* DRIVE STATUS */
        int     bder;           /* ERROR REGISTER */
        int     bdcs;           /* COMMAND & STATUS */
        int     bdbc;           /* BYTE COUNT */
        caddr_t bdba;           /* BUS ADDRESS */
        int     bdca;           /* CYLINDER ADDRESS */
        int     bdda;           /* DISK ADDRESS (HEAD+SECTOR) */
        char    bdea;           /* FOR EXTENDED ADDRESSING */
};

           /* DEVICE DEPENDENT */

#define BDADDR ((struct device *) 00)      /* DEVICE REGISTER ADDRESS */

      /*    DEFINITIONS OF BDCS BITS    */
         /*    DEVICE DEPENDENT   */
```

```
#define GO        0000001          /* EXECUTE COMMAND */
#define RESET     0000000          /* RESET CONTROLLER */
#define SKCMD     0000006          /* SEEK COMMAND */
#define RDCMD     0000004          /* READ COMMAND */
#define WTCMD     0000002          /* WRITE COMMAND */
#define IENABLE   0000100          /* INTERRUPT ENABLE */
#define READY     0000200          /* CONTROLLER READY */
#define ERR       0100000          /* ERROR */

      /*    DEFINITION OF BDDS BITS     */
          /*   DEVICE DEPENDENT    */


#define SKCMPL    0000100          /* SEEK COMPLETED */

#define SPL()     spl5()           /* DEVICE PRIORITY LEVEL */


        /*  DEVICE PARAMETERS  */
          /* DEVICE DEPENDENT */

#define NBD       4                /* NUMBER OF DRIVES */
#define NCYL      150              /* NUMBER OF CYLINDERS */
#define NBPC      100              /* NUMBER OF BLOCKS PER CYLINDER */
#define NSPT      10               /* NUMBER OF SECTORS PER TRACK */
#define ROOTSZ    ((4999/NBPC)+1)  /* SIZE OF ROOT FILSYS AREA (IN CLYINDERS) */
#define SWAPSZ    ((1999/NBPC)+1)  /* SIZE OF SWAP AREA (IN CYLINDERS) */
#define USROFS    (ROOTSZ+SWAPSZ)  /* BEGINNING OF USR AREA (IN CYLINDERS) */
#define USERSZ    (NCYL-USROFS)    /* SIZE OF USER FILSYS AREA (IN CYLINDERS) */
#define NCYL2     (NCYL/2)

            /* TABLE OF PSUEDO-DISK SIZES */

struct {
        daddr_t nblocks;           /* NUMBER OF BLOCKS IN PARTITION */
        int     cyloff;            /* OFFSET TO PARTITION IN CYLINDERS */
} bd_sizes[8] = {                  /* NUMBER OF PARTITIONS ON DEVICE */
        NCYL*NBPC,      0,         /* CYL 0 THRU END OF DISK       */
        ROOTSZ*NBPC,    0,         /* ROOT AREA (ABOUT 5000 BLOCKS)        */
        SWAPSZ*NBPC,    ROOTSZ,    /* SWAP AREA (ABOUT 2000 BLOCKS)        */
        USERSZ*NBPC,    USROFS,    /* USR AREA (REST OF DISK)              */
        0,      0,                 /* SPARE                                */
        0,      0,                 /* SPARE                                */
        NCYL2*NBPC,     0,         /* FIRST HALF OF THE DISK               */
        NCYL2*NBPC,     NCYL2,     /* LAST HALF OF THE DISK                */
};

              /*   STRUCTURES USED   */

struct  buf     bdtab;             /* START OF BUFFER QUEUE  */
struct  buf     rbdbuf;            /* USED FOR RAW I/O */
```

```
/*
 * MONITORING DEVICE NUMBER
 */
#define DK_N    0

/*
 *    Strategy Routine:
 *    Arguments:
 *      Pointer to buffer structure
 *    Function:
 *      Check validity of request
 *      Queue the request
 *      Start-up the device if idle
 *
 */

bostrategy(bp)
register struct buf *bp;            /* POINTER TO BUFFER STRUCTURE */
{
        register struct buf *dp;
        register int unit;          /* UNIT NUMBER */
        long sz;

        unit = minor(bp->b_dev);          /* COMPUTE PARTITION DESIRED */
        sz = bp->b_bcount;                /* COMPUTE NUMBER OF CHARACTERS */
        sz = (sz+BMASK)>>BSHIFT;
        if (unit >= (NBD<<3) ||       /* CHECK FOR DEV/BLK OUT OF RANGE */
            bp->b_blkno+sz >= bd_sizes[unit&07].nblocks) {
                if (bo->b_flags & B_READ && unit < (NBD<<3))
                        bp->b_resid = bp->b_count;  /* CHECK FOR EOF */
                else
                        bp->b_flags |= B_ERROR;     /* SET ERROR BIT */
                iodone(bp);                   /* LET OTHERS KNOW IT IS FREE */
                return;
        }
        bp->av_forw = NULL;         /* TAKE OFF FREE LIST */

/* Here we add the buffer to the queue of requests; lock out
interrupts while updating queue pointers; and start the
device if it is not active.  Requests are sorted by cylinder
number by the routine 'disksort'.  This will minimize seek
time and make the I/O more efficient.  The field 'b_cylin'
is not in the buf.h structure of V2.2A of XENIX.  It will be
in all subsequent XENIX releases.  To compensate for its
omission, simply remove the comment delimiters from the
following statement.
 */
```

```
/*#define b_cylin b_resid          */  /* DEPENDS ON VERSION NUMBER */

        bp->b_cylin = b_blkno/NBPC+(bd_sizes[unit]&07)  /* CYLINDER NUMBER */
        dp = &bdtab                     /* SET DEVICE POINTER TO BUFFER */
        unit = SPL();                   /* LOCK OUT INTERRUPTS */
        disksort(dp,bp)                 /* SORT REQUESTS */
        if (dp->b_active == NULL)       /* START UP THE DEVICE */
                bdstart();
        splx(unit);                     /* ALLOW INTERRUPTS */
}

/*
 *      Startup Routine:
 *      Arguments:
 *        None
 *      Function:
 *        Compute device dependent parameters
 *        Start-up device
 *        Indicate request to I/O monitor routines
 */
bdstart()
{
        register struct buf *bp;        /* BUFFER POINTER */
        register int unit;              /* MINOR DEVICE NUMBER */
        int cmd,cn,tn,sn,dn;            /* I/O PARAMETERS */
        daddr_t bn;                     /* DISK ADDRESS OF BUFFER */


        if ((bp = bdtab.b_actf) == NULL)      /* QUEUE IS EMPTY */
                return;
        bdtab.b_active++;                     /* ACTIVATE DEVICE */
/*
 *      Compute parameters for device registers:
 *              dn=drive number,
 *              bn=block number,
 *              cn=cylinder number
 *              tn=track number,
 *              sn=sector number,
 *              cmd=command.
 */
        unit = minor(bp->b_dev);
        dn = unit>>3;
        bn = bp->b_blkno;
        cn = bn->b_cylin;
        sn = bn%(NBPC);
        tn = sn/(NSPT);
        sn = sn%(NSPT);

        cmd =  IENABLE | GO;
        if (bp->b_flags & B_READ)             /* IF ITS A READ */
                cmd |= RDCMD; else            /* SET CMD = READ */
```

```
                cmd |= WTCMD;                   /* ELSE CMD= WRITE */

/*
 *      Write to device registers
 */
        out(&BDADDR->bdca, cn);                 /* CYLINDER ADDRESS */
        out(&BDADDR->bdda, (tn<<4) | sn);       /* DISK ADDRESS */
        out(&BDADDR->bdba, bp->b_un.b_addr);    /* BUS ADDRESS */
        out(&BDADDR->bdea, bp->xmem);           /* HIGH ORDER CORE ADDRESS */
        out(&BDADDR->bdbc, bp->b_bcount);       /* BYTE COUNT */
        out(&BDADDR->bdcs, (dn<<8) | cmd);      /* COMMAND AND STATUS */

/*
 *      THIS SECTION OF CODE IS OPTIONAL FOR PERFORMANCE MONITORING ROUTINES
 *
 *      Set up i/o monitor routines
 */

#ifdef DK_N
        dk_busy |= 1<<DK_N;
        dk_numb[DK_N] += 1;
        unit = bp->b_bcount>>6;
        dk_wds[DK_N] += unit;
#endif
}

/*
 *      Interrupt routine:
 *        Check completion status
 *        Indicate completion to i/o monitor routines
 *        Log errors
 *        Re-start (on error) or start next request
 */
bdintr()
{
        register struct buf *bp;

        if (bdtab.b_active == 0)            /* IGNORE SPURIOUS INTRPT */
                return;
#ifdef DK_N
        dk_busy &=  ~(1<<DK_N);
#endif
        (bp = bdtab.b_actf);
        bdtab.b_active = 0;
        if (in(&BDADDR->bdcs) & ERR) {              /* error bit */
                deverror(bp, in(&BDADDR->bder), in(&BDADDR->bdds));
                out(&BDADDR->bdcs, RESET|GO);
                if (++bdtab.b_errcnt <= 10) {
                        bdstart();
                        return;
                }
```

```
                bp->b_flags |= B_ERROR;
        }
/*
 *      Flag current request complete, start next one
 */
        bdtab.b_errcnt = 0;
        bdtab.b_actf = bp->av_forw;
        bp->b_resid = 0;
        iodone(bp);
        bdstart();
}

/*
 *      raw read routine:
 *      This routine calls 'physio' which computes and validates a physical
 *      address from the current logical address.
 *
 *      Arguments
 *        Full Device Number
 *      Functions:
 *        Call physio which do the actual raw (physical) I/O
 *        The arguments to physio are:
 *          pointer to the strategy routine
 *          buffer for raw I/O
 *          device
 *          read/write flag
 */

bdread(dev)
{

        physio(bdstrategy, &rbdbuf, dev, B_READ);
}

/*
 *       Raw write routine:
 *       Arguments(to bdwrite):
 *         Full Device Number
 *       Functions:
 *         Call physio which will perform the actual raw (physical) I/O
 *
 *      The routine  'physio' computes and validates a physical
 *      address from the current logical address.
 *
 *        The arguments to physio are:
 *          pointer to the strategy routine
 *          buffer for raw I/O
 *          device
 *          read/write flag
 *
 */
```

```
bdwrite(dev)
{

        physio(bdstrategy, &rbdbuf, dev, B_WRITE);
}
```

A.2 Prototype tape driver

This is a prototype of a magnetic tape driver. The logical addressing is set up so that the 0200 bit of the minor number indicates that the device should not be rewound on close. The 0100 bit is used to indicate high-density. The remaining bits of the minor device number are used to address the physical drive number. Some of these bits could also be used to address a tape controller. The driver is set up so that commands to the drive are sent through a special buffer header. Provisions are made to interlock when required.

```
 */

#include "../h/param.h"          /* SYSTEM PARAMETERS */
#include "../h/buf.h"            /* BUFFER STRUCTURE */
#include "../h/dir.h"            /* DIRECTORY STRUCTURE */
#include "../h/conf.h"           /* BLOCK DEVICE DECLARATIONS */
#include "../h/file.h"           /* FILE STRUCTURE */
#include "../h/user.h"           /* USER STRUCTURE */

         /* STRUCTURE OF DEVICE REGISTERS */
             /*  DEVICE DEPENDENT */

struct device {
        int     btds;            /* DRIVE STATUS */
        int     btcs;            /* COMMAND & STATUS */
        int     btbc;            /* BYTE/RECORD COUNT */
        caddr_t btba;            /* BUS ADDRESS */
        char    btea;            /* EXTENDED ADDRESSING */
};

             /* DEVICE DEPENDENT */

#define BTADDR ((struct device *)00)  /* ADDRESS OF DEVICE REGISTERS */
#define SPL()   spl5()                /* INTERRUPT LEVEL OF DEVICE */


        /*   DEFINITIONS OF BITS IN BTCS     */
              /* DEVICE DEPENDENT */

#define GO      0000001
#define RCOM    0000002          /* READ CMD */
#define WCOM    0000004          /* WRITE CMD */
```

```
#define WEOF     0000006          /* WRITE END OF FILE */
#define SFORW    0000010          /* SPACE FORWARD CMD */
#define SREV     0000012          /* SPACE REVERSE CMD */
#define WIRG     0000014          /* WRITE INTER-RECORD-GAP */
#define REW      0000016          /* REWIND CMD */
#define NOP      0000200          /* NO OPERATION */
#define IENABLE  0000100          /* INTERRUPT ENABLE */
#define CRDY     0000200          /* CONTROLLER READY */
#define DENS16   0040000          /* 1600 BPI */
#define DENS8    0000000          /* 800 BPI */
#define ERR      0100000          /* ERROR */

            /* DEFINITIONS OF BITS IN BTDS */
               /* DEVICE DEPENDENT */

#define TUR      0000001          /* TAPE UNIT READY */
#define WL       0000004          /* WRITE LOCKED */
#define RLE      0001000          /* RECORD LENGTH ERROR */
#define EOF      0040000          /* END OF FILE */
#define HARD     0102200          /* HARD ERROR BITS: ILC, EOT */

                 /* LOCAL STRUCTURES USED */

struct  buf      bttab;           /* HEAD OF REQUEST QUEUE */
struct  buf      rbtbuf;          /* BUFFER FOR RAW I/O REQUESTS */
struct  buf      cbtbuf;          /* COMMAND BUFFER */


#define NBT      4                /* NUMBER OF DRIVES */


            /* THE FOLLOWING ARRAYS ARE EACH COMPRISED OF 4
                 ELEMENTS, ONE FOR EACH POTENTIAL DRIVE  */

char     t_flags[NBT];            /* PER-DEVICE FLAGS */
daddr_t  t_blkno[NBT];            /* ADDRESS OF BLOCK */
daddr_t  t_nxrec[NBT];            /* ADDRESS OF NEXT RECORD */
int      t_dens [NBT];            /* TAPE DENSITY */

            /* DEFINITIONS OF DRIVER STATES */

#define SSEEK    1                /* EXECUTING SPACING COMMAND */
#define SIO      2                /* EXECUTING I/O COMMAND */
#define SCOM     3                /* EXECUTING COMMAND BUFFER */

          /* DEFINITIONS OF FLAG BITS */

#define T_WROTE  1                /* UNIT HAS BEEN WRITTEN ON */
#define T_OPEN   2                /* UNIT IS OPEN */
#define T_ERR    4                /* UNIT HAS HAD HARD ERROR */
```

```
/*
 *      Open Routine
 *      Arguments:
 *        Device number
 *        Read/write flag
 *      Functions:
 *        Check unit number
 *        Insure one open per unit
 *        Setup the device parameters
 */

btopen(dev, flag)
{
        register unit, ds;

        unit = minor(dev) & 077;                /* CALCULATE UNIT NUMBER */
        if (unit >= NBT || t_flags[unit]&T_OPEN) { /* VALID UNIT ?? */
                u.u_error = ENXIO;
                return;
        }
        t_blkno[unit] = 0;                      /* SET BLOCK NUMBER TO ZERO */
        t_nxrec[unit] = 1000000L;               /* SET SEEKING RANGE */
        t_dens[unit] = minor(dev)&0100? DENS16: DENS8;      /* DENSITY */

        bttab.b_flags |= B_TAPE;                /* THIS IS A MAGTAPE */
        ds = tcommand(dev, NOP);                /* DETERMINE DRIVE STATUS */
        if ((ds&TUR)==0) {                      /* IF DRIVE IS NOT READY */
                printf("mt%d off line\n",unit);
                u.u_error = ENXIO;
        }
        if (flag && ds&WL) {                    /* CHECK WRITE PROTECTION */
                printf("mt%d needs write ring\n",unit);
                u.u_error = ENXIO;
        }
        if (u.u_error==0)                       /* IF NO ERROR, OPEN UNIT */
                t_flags[unit] = T_OPEN; /* SET FLAG */
}

/*
 *      Close Routine
 *      Arguments:
 *        Device number
 *        Flag
 *      Functions:
 *        If writing, write end of tape
 *        If a rewind device, rewind and
 *         allow opens
 */

btclose(dev, flag)
dev_t dev;
```

```
int flag;
{

        if ( flag == FWRITE ||
        ((flag&FWRITE) && (t_flags[minor(dev)&077]&T_WROTE))) {
                tcommand(dev, WEOF);            /* WRITE END OF TAPE MARK */
                tcommand(dev, WEOF);
                tcommand(dev, SREV);            /* SPACE TO BETWEEN EOFs */
        }
        if ((minor(dev)&0200) == 0)             /* IS TAPE IS TO BE REWOUND */
                tcommand(dev, REW);             /* YES, REWIND TAPE */
        t_flags[minor(dev)&077] = 0;            /* CLEAR OUT FLAG BITS */
}

/*
 *      Command Routine
 *      Arguments:
 *        Device number
 *        Device command
 *      Functions:
 *        Wait until device quiescent
 *        Send requested command to device
 */
tcommand(dev, cmd)
{
        register struct buf *bp;
        register ps;

        bp = &cbtbuf;                           /* ASSIGN COMMAND BUFFER */
        ps = SPL();                             /* LOCK OUT INTERRUPTS */
        while (bp->b_flags&B_BUSY) {            /* WHILE COMMAND BUFFER IS BUSY */
                bp->b_flags |= B_WANTED;        /* INDICATE IT'S WANTED */
                sleep((caddr_t)bp, PRIBIO);     /* SLEEP UNTIL READY */
        }
        bp->b_flags = B_BUSY|B_READ;        /* INDICATE IT IS BUSY READING*/
        splx(ps);                               /* ALLOW INTERRUPTS */
        bp->b_dev = dev;                        /* SET DEVICE TO COMMAND BUFFER */
        bp->b_resid = cmd;                      /* SET COMMAND */
        bp->b_blkno = 0;                        /* SET BLOCK NUMBER */
        btstrategy(bp);                         /* CALL STRATEGY ROUTINE */
        iowait(bp);                             /* WAIT UNTIL I/O IS DONE */
        if (bp->b_flags&B_WANTED) {             /* IF DEVICE IS WANTED */
                bp->b_flags&= ~(B_WANTED|B_BUSY)
                wakeup((caddr_t)bp);            /* WAKEUP */
        }
        return(bp->b_resid);                    /* RETURN DRIVE STATUS */
}


/*
 *      Strategy Routine: Entry point to driver proper
```

```
 *      Arguments:
 *        Pointer to Buffer
 *      Functions:
 *        Queue Requests
 */

btstrategy(bp)
register struct buf *bp;
{
        register daddr_t *p;
        register unit;                                  /* UNIT NUMBER */

        if (bp != &chtbuf) {                            /* IF NOT A COMMAND */
                unit = minor(bp->b_dev)&077;            /* GET MINOR DEVICE NUMBER */
                p = &t_nxrec[unit];                     /* MAXIMUM RECORD NUMBER */
                if (*p <= bp->b_blkno) {                /* RANGE CHECKING */

                        if (*p < bp->b_blkno) {/* OUT OF RANGE */
                                bp->b_flags |= B_ERROR;     /* SET ERROR */
                                iodone(bp);             /* INDICATE COMPLETION */
                                return;
                        }
                        if (bp->b_flags&B_READ) {       /* IF READING */
                                clrbuf(bp);             /* CLEAR BUFFER */
                                bp->b_resid = 0;        /* CLEAR COMMAND */
                                iodone(bp);             /* INDICATE COMPLETION */
                                return;                 /* RETURN */
                        }
                }
                if ((bp->b_flags&B_READ) == 0) {        /* IF IT IS A WRITE */
                        t_flags[unit] |= T_WROTE;       /* SET FLAG */
                        *p = bp->b_blkno+1;             /* NEW MAX. RECORD */
                }
        }
        bp->av_forw = 0;
        unit = SPL();                                   /* IGNORE INTERRUPTS */
        if (bttab.b_actf == NULL)                       /* QUEUE REQUEST */
                bttab.b_actf = bp;
        else
                bttab.b_actl->av_forw = bp;
        bttab.b_actl = bp;
        if (bttab.b_active == 0)                        /* IF DEVICE NOT ACTIVE */
                btstart();                              /* START DEVICE */
        splx(unit);                                     /* ALLOW INTERRUPTS */
}

/*
 *      Startup Routine: Initiate a request
 *      Arguments:
 *        None
 *      Functions:
```

```
 *         If buffer ptr is command buf, execute command
 *          else execute read/write
 */

btstart()
{
        register struct buf *bp;                        /* POINTER TO BUFFER */
        register int cmd;                               /* COMMAND */
        int unit;                                       /* DEVICE NUMBER */
        register daddr_t *blkno;                        /* BLOCK ADDRESS */

loop:
        if ((bp = bttab.b_actf) == NULL)                /* QUEUE IS EMPTY */
                return;
        unit = minor(bp->b_dev)&077;                    /* GET DEVICE NUMBER */
        blkno = &t_blkno[unit];                         /* GET CURRENT BLOCK NUMBER */
        if (t_flags[unit]&T_ERR || (in(&BTADDR->btcs)&CRDY)==0) {
                bp->b_flags |= B_ERROR;
                goto next;
        }

                /* Set up common part of command */
                     /* DEVICE DEPENDENT */

        cmd = t_dens[unit] | (unit<<8) | IENABLE;

/*
 *      Execute tape commands as specified in the buffer
 */

        if (bp == &cbtbuf) {                            /* IF IT'S THE COMMAND BUFER */
                if (bp->b_resid == NOP) {               /* AND THE COMMAND = NOP */
                        bp->b_resid = in(&BTADDR->btds);  /* READ DRIVE STATUS */
                        goto next;                      /* CONTINUE */
                }
                bttab.b_active = SCOM;                  /* START EXECUTING CMD BUFFER */

        /* Determine Command and Write Control&Status register */

                         /* DEVICE DEPENDENT */

                cmd |=  bp->b_resid | GO
                out(&BTADDR->btcs, cmd);

                return;
        }


/*
 *      Determine if a spacing command is required
 *      If it is, then:
```

```
 *                  Set state to SSEEK
 *                  Give spacing Command
 */

        if (*blkno != bp->b_blkno) {                    /* IF NOT CORRECT BLOCK */
                bttab.b_active = SSEEK;                 /* SET STATE TO SEEK */
                if (*blkno < bp->b_blkno) {             /* DETERMINE DIRECTION */
                        cmd |= SFORW|GO;                /* FORWARD SEEK */

                  /* Write Record Count Register  DEVDEP */
                        out(&BTADDR->btbc, (int) (bp->b_blkno-*blkno));

                } else {
                        if (bp->b_blkno == 0)           /* AT BEGINNING */
                                cmd |= REW|GO;          /* REWIND */
                        else {
                                cmd |= SREV|GO;         /* REVERSE WIND */

                    /* Update Record Count Register  DEVDEP */
                                out(&BTADDR->btbc, (int) (*blkno-bp->b_blkno));
                        }
                }
                out(&BTADDR->btcs, cmd);                /* WRITE THE C&S REGISTER */
                return;
        }

        /* No seeking was required; start I/O and update registers */

                /* DEVICE DEPENDENT */

        bttab.b_active = SIO;
        out(&BTADDR->btba, bp->b_un.b_addr);
        out(&BTADDR->btea, bp->b_xmem);             /* HIGH ORDER CORE ADDRESS */
        out(&BTADDR->btbc, bp->b_bcount);
        out(&BTADDR->btcs,
            cmd | ((bp->b_flags&B_READ)? RCOM|GO:
            ((bttab.b_errcnt)? WIRG|GO: WCOM|GO)));
        return;

next:
        bttab.b_actf = bp->av_forw;
        bttab.b_active = 0;
        iodone(bp);                                 /* ALERT OTHER PROCESSES */
        goto loop;
}

/*
 *      Interrupt Routine:
 *      Arguments:
 *        None
 *      Functions:
```

```
 *          Log Errors
 *          Restart controller

 */
btintr()
{
        register struct buf *bp;                        /* POINTER TO BUFFER */
        register int unit, err;
        int     state;

        if ((bp = bttab.b_actf) == NULL)
                return;
        unit = minor(bp->b_dev)&077;                    /* GET DEVICE NUMBER */
        state = bttab.b_active;                         /* SAVE STATE OF DEVICE */
        bttab.b_active = 0;                             /* REINITIALIZE THE STATE */
        if (in(&BTADDR->btcs) & ERR) {                  /* CHECK FOR ERROR */
                err = in(&BTADDR->btds);                /* DETERMINE TYPE */
                if (err&EOF) {                          /* END OF FILE */
                        t_nxrec[unit] = bp->b_blkno;
                        state = SCOM;
                        out(&BTADDR->btbc, bp->b_bcount);
                        goto ret;
                }
                if ((err&HARD) == 0 && err&RLE) {  /* RECORD LENGTH ERROR */
                        state = SIO;                    /* EXECUTING I/O */
                        goto ret;
                }
                if ((err&(HARD|EOF)) == 0 && state==SIO) { /* EXECUTING I/O */
                        if (++bttab.b_errcnt < 3) {
                                t_blkno[unit]++;
                                bttab.b_active = 0;
                                btstart();
                                return;
                        }
                } else
                        if ((t_flags[unit]&T_ERR)==0 && bp!=&rbtbuf &&
                                (err&EOF)==0 ) {
                                t_flags[unit] |= T_ERR;
                                deverror(bp, err, in(&BTADDR->btcs));
                        }
                bp->b_flags |= B_ERROR;
                state = SIO;
        }
ret:
        switch ( state ) {
        case SIO:
                t_blkno[unit] += (bp->b_bcount>>BSHIFT);
        case SCOM:
                bttab.b_errcnt = 0;
                bttab.b_actf = bp->av_forw;
                bp->b_resid = in(&BTADDR->btbc);   /* WRITE BYTE/RECORD REG */
```

```
                iodone(bp);                             /* ALERT OTHER PROCESSES */
                break;
        case SSEEK:
                t_blkno[unit] = bp->b_blkno;
                break;
        default:
                return;
        }
        btstart();
}

/*
 *      Btread
 *      Arguments:
 *         Device Number
 *      Functions:
 *         Determine block address on device
 *         Call the routine to do the acutal physical I/O
 *
 *      'physio' is the routine which performs the actual physical I/O
 *       from the current logical address;  essentially, all the work
 *       of physio is computing physical and validating physical addresses.
 */

btread(dev)
{
        btphys(dev);                                    /* CALCULATE BLOCK ADDRESS */
        physio(btstrategy, &rbtbuf, dev, B_READ);       /* DO RAW I/O */
}

/*
 *      Btwrite
 *      Arguments:
 *        Device number
 *      Functions:
 *        Determine block address on device
 *        Perform actual physical I/O
 *
 *      'physio' is the routine which performs the actual physical I/O
 *       from the current logical address;  essentially, all the work
 *       of physio is computing physical and validating physical addresses.
 */

btwrite(dev)
{
        btphys(dev);                                    /* GET BLOCK ADDRESS */
        physio(btstrategy, &rbtbuf, dev, B_WRITE);      /* DO RAW I/O */
}


/*      Btphys
```

```
 *       Arguments:
 *         Device number
 *       Functions:
 *         Calculate block address on device
 */

btphys(dev)
{
        register unit;
        daddr_t a;

        unit = minor(dev) & 077;                /* GET DEVICE NUMBER */
        if(unit < NBT) {
                a = u.u_offset >> BSHIFT;       /* COMPUTE BLOCK NUMBER */
                t_blkno[unit] = a;              /* DISALLOW SEEKS ON */
                t_nxrec[unit] = a+1;            /* RAW DEVICES */
        }
}
```

APPENDIX B:

Games

Included here are the games available on the XENIX system.

NAME

arithmetic - provide drill in number facts

SYNOPSIS

/usr/games/arithmetic [**+-x/**] [range]

DESCRIPTION

Arithmetic types out simple arithmetic problems, and waits for an answer to be typed in. If the answer is correct, it types back "Right!", and a new problem. If the answer is wrong, it replies "What?", and waits for another answer. Every twenty problems, it publishes statistics on correctness and the time required to answer.

To quit the program, type an interrupt (delete).

The first optional argument determines the kind of problem to be generated; **+-x/** respectively cause addition, subtraction, multiplication, and division problems to be generated. One or more characters can be given; if more than one is given, the different types of problems will be mixed in random order; default is **+-**

Range is a decimal number; all addends, subtrahends, differences, multiplicands, divisors, and quotients will be less than or equal to the value of range. Default range is 10.

At the start, all numbers less than or equal to range are equally likely to appear. If the respondent makes a mistake, the numbers in the problem which was missed become more likely to reappear.

As a matter of educational philosophy, the program will not give correct answers, since the learner should, in principle, be able to calculate them. Thus the program is intended to provide drill for someone just past the first learning stage, not to teach number facts de novo. For almost all users, the relevant statistic should be time per problem, not percent correct.

NAME

backgammon - the game

SYNOPSIS

/usr/games/backgammon

DESCRIPTION

This program does what you expect. It will ask whether you need instructions.

NAME

quiz - test your knowledge

SYNOPSIS

/usr/games/quiz [-i file] [-t] [category1 category2]

DESCRIPTION

Quiz gives associative knowledge tests on various subjects. It asks items chosen from category1 and expects answers from category2. If no categories are specified, quiz gives instructions and lists the available categories.

Quiz tells a correct answer whenever you type a bare newline. At the end of input, upon interrupt, or when questions run out, quiz reports a score and terminates.

The -t flag specifies `tutorial' mode, where missed questions are repeated later, and material is gradually introduced as you learn.

The -i flag causes the named file to be substituted for the default index file. The lines of these files have the syntax:

```
line      = category newline | category `:' line
category  = alternate | category `|' alternate
alternate = empty | alternate primary
primary   = character | `[' category `]' | option
option    = `{' category `}'
```

The first category on each line of an index file names an information file. The remaining categories specify the order and contents of the data in each line of the information file. Information files have the same syntax. Backslash `\' is used as with sh(1) to quote syntactically significant characters or to insert transparent newlines into a line. When either a question or its answer is empty, quiz will refrain from asking it.

FILES

/usr/games/quiz.k/*

BUGS

The construct `a|ab' doesn't work in an information file. Use `a{b}'.

NAME

hangman, words - word games

SYNOPSIS

/usr/games/hangman [dict]

/usr/games/words

DESCRIPTION

Hangman chooses a word at least seven letters long from a word list. The user is to guess letters one at a time.

The optional argument names an alternate word list. The special name `-a' gets a particular very large word list.

Words prints all the uncapitalized words in the word list that can be made from letters in string.

FILES

/usr/dict/words the regular word list
/crp/dict/web2 the the -a word list

DIAGNOSTICS

After each round, hangman reports the average number of guesses per round and the number of rounds.

BUGS

Hyphenated compounds are run together.

UNIX software is distributed without the -a word list.

NAME

wump - the game of hunt-the-wumpus

SYNOPSIS

/usr/games/wump

DESCRIPTION

Wump plays the game of `Hunt the Wumpus.' A Wumpus is a creature that lives in a cave with several rooms connected by tunnels. You wander among the rooms, trying to shoot the Wumpus with an arrow, meanwhile avoiding being eaten by the Wumpus and falling into Bottomless Pits. There are also Super Bats which are likely to pick you up and drop you in some random room.

The program asks various questions which you answer one per line; it will give a more detailed description if you want.

This program is based on one described in People's Computer Company, 2, 2 (November 1973).

BUGS

It will never replace Space War.