UNOS 1320

Bell Laboratories

subject: **Setting Up UNIX/TS**

date: **September 30, 1978**

from: **R. C. Haight**
MH 8234
2F211 x7498
MF 78-8234-98

L. A. Wehr
MH 8234
2F245 x4896
MF 78-8234-98

MEMORANDUM FOR FILE

The attached document describes programming steps for generating a UNIX/TS operating system along with administrative detail on configuration, setting up file systems, and installation/recompilation of command software.

R. C. Haight

MH-8234-RCH/LAW

L. A. Wehr

Setting Up UNIX/TS

R. C. Haight

Bell Laboratories
Murray Hill, New Jersey 07974

L. A. Wehr

Bell Laboratories
Murray Hill, New Jersey 07974

1. INTRODUCTION

1.1 Prerequisites

Before attempting to generate a UNIX/TS system, you should understand that a considerable knowledge of the attendant documentation is required and assumed. In particular, you should be very familiar with the following documents:

- *The UNIX Time-Sharing System*
- *UNIX/TS User's Manual*
- *C Reference Manual*
- *Administrative Advice for UNIX/TS (to be issued)*
- *UNIX/TS Operations Manual (to be issued)*

A complete list of pertinent documentation is given in *Documents for UNIX/TS* (to be issued). Also, throughout this document, each reference of the form *name*(N), where N is a Arabic number, refers to page *name* in Section N of the *UNIX/TS User's Manual.*

You must have a basic understanding of the operation of the hardware. This includes the console panel, the tape drives, and the disk drives, all of which are assumed to have standard addresses and interrupt vectors. If you do not understand something, consult the *UNIX/TS Operations Manual* and DEC® maintenance manuals. It is also assumed that the hardware works and has been completely installed. The DEC diagnostics should have been run to test the configuration, and you must have a detailed description of the hardware, including device addresses, interrupt vectors, and bus levels. This information is very important to generate the UNIX/TS system.

1.2 Procedure

UNIX/TS is distributed on a single, multi-file magnetic tape, recorded 9-track at 800bpi. The initial load program will copy a file system from tape (either a TU10 or a TU16) to disk (either an RP03 or an RP06). Note that RP04 and RP05 drives are considered to be equivalent to RP06 drives; any differences will be noted explicitly. Once the *root* file system has been successfully loaded to disk, UNIX/TS may be booted and the available utility programs may then be used to complete the installation.

The remaining files on the tape contain source text and supplemental commands. These files contain essential information to generate a new system that will match your particular hardware and software environment.

In order for any of the update procedures to work correctly, you *must* be *running* UNIX/TS. Older versions of UNIX cannot be correctly *updated* with a UNIX/TS system. The *cpio*(1) program will not replace any file if its replacement has a modification time that is less than (i.e., earlier than) the modification time of the original file. This can be due to local modifications. Furthermore, certain administrative files (e.g., *passwd, crontab*) are sent with modification times of Jan 1, 1970 to ensure that they do not replace their counterparts during updates. Any file not copied will cause *cpio*(1) to print a message to that effect. These messages should always be investigated to ensure that any files not copied were of that type. However, note that, depending on respective modification times, a locally-modified file may get updated, thus destroying the

local modifications.

There are several difficulties that can arise when installing a UNIX/TS system. One of the most common problems is running out of disk space when performing an update. Should this occur, the original contents of the file system should be restored from a backup copy and the contents of the update tape should be read into a spare file system using the *cpio*(1) program. Unwanted material can then be removed and the original file system can be updated from this new file system using the −p option of *cpio*(1). Modification times of files should also be preserved using the −m option of *cpio*(1).

This document is not strictly linear. Read it thoroughly, from start to finish, and then read it again. Also, remove the write ring, if present, from the distribution tape to guard against accidental erasure.

2. LOAD PROCEDURES

2.1 Distribution Tape Format

The five files are: a loader, a physical copy of the *root* file system, the *cpio*(1) program, and a *cpio*(1) structured copy of the *root* file system, and a (*cpio*) copy of */usr*. *Root* refers to the directory "/", which is the root of all the directory trees. The format of this tape is as follows:

record 0:	Tape boot loader−512-bytes;
record 1:	Tape boot loader−512-bytes;
remainder of file 1:	Initial load program−several 512-byte records;
end-of-file.	
file 2:	*root* file system (physical)−several 5120-byte records (blocking factor 10);
end-of-file.	
file 3:	*cpio* program (latest version)−several 512-byte records;
end-of-file.	
file 4:	*root* file system (structured in *cpio* format)−several 5120-byte records (to be used *only* for updating an earlier UNIX/TS);
end-of-file.	
file 5:	*/usr* file system (same format as file 4).
end-of-file.	

The *root* (/) file system contains the following directories:

bck:	Directory used to mount a backup file system for file restoral.
bin:	Public commands; most of what's described in Section 1 of the *UNIX/TS User's Manual.*
dev:	Special files, all the devices on the system.
etc:	Administrative programs and tables.
lib:	Public libraries, parts of the assembler, C compiler.
mnt:	Directory used to mount a file system.
stand:	Standalone boot programs.
tmp:	Directory used for temporary files; should be cleaned at reboot.
usr:	Directory used to mount the */usr* file system; user directories often kept here also.

2.2 Initial Load of root

Mount the distribution tape on drive 0 and position it at the load point. Next, bootstrap the tape by reading either record 0 or record 1 into memory starting at address 0 and start execution at address 0. This may be accomplished by using a standard DEC ROM bootstrap loader, a special ROM, or some manual procedure. See *romboot*(8), *tapeboot*(8), and *70boot*(8).

The tape boot loader will then type "UNIX tape boot loader" on the console terminal and read in and execute the initial load program. The program will then type detailed instructions about the operation of the program on the console terminal. First, it will ask what type of disk drive you have and which drive you plan to use for the copy. The disk controller used must be at the standard DEC address indicated by the program. However, other disk controllers on your system may be at non-standard addresses. You must mount a formatted, error-free pack on the drive you have indicated. If necessary, use the appropriate DEC diagnostic program to format the pack. Note that the pack will be written on. Second, the program will ask what type of tape drive you have and which drive contains the tape. Normally, this will be drive 0, but the program will work with other drives. Note that the tape is currently positioned correctly after the end-of-file between the initial load program and the *root* file system. When everything is ready, the program will copy the file system from the tape to disk and give instructions for booting UNIX/TS. After the copy is complete and you have booted the basic version of UNIX/TS, check (using *fsck*(1M)) the *root* file system and browse through it.

The file */stand/mmtest* is a stand-alone memory mapping diagnostic program. It should be booted and run (20 minutes) if you are not *absolutely* sure that DEC FCO (field change order) M8140-R002 has been applied to your PDP 11/70 CPU!

2.3 Update of root

It is very important that the system be running in *single-user* mode during the update phase. To update an already existing *root* file system, files three and four will be used. It is necessary to first make a copy of your *root* file system using *volcopy*(1M) and then update this copy. The copy should be made on a separate disk pack using the same section number as your *root* file system (always section 0). Also, after the update is completed, check if any of your local administrative files in the directory */etc* need modification. Most of these are mentioned in Section 4 below.

Mount the tape on drive 0 and position it at the load point. We assume that disk drive 1 is available for making the copy, and that the *root* file system is on */dev/rp0*. The following procedure will first make a copy of the *root* file system, and then update this copy. Note that */dev/mt4* refers to tape drive 0 but has the side effect of spacing forward to the next end-of-file (no rewind option). The *B* option of *cpio* specifies that input is in 5120-byte records.

```
volcopy root /dev/rrp0 pkname1 /dev/rrp10 pkname2
mount /dev/rp10 /bck
: The two echo's will move the tape to file 3.
echo <⁄dev/mt4
echo <⁄dev/mt4
cp /dev/mt4 /bck/bin/cpio
chmod 755 /bck/bin/cpio
chown bin /bck/bin/cpio
cd /bck
/bck/bin/cpio -idmB </dev/rmt0
cd /
umount /dev/rp10
```

where *pkname1* and *pkname2* are the volume names of the source and destination disk packs, respectively. If the new copy is satisfactory, shut down and halt the system, change disk packs, and reboot the system using the new *root*.

2.4 File 5 (/usr) Format

File 5 contains the */usr* file system in *cpio*(1) format (5120-byte records). The */usr* file system contains commands and files that must be available (mounted) when the system is in *multi-user* mode. The tape contains the following directories:

adm: Miscellaneous administrative command and data files, including the connect-time accounting file *wtmp* and the process accounting file *pacct*.

bin:	Public commands; an overflow for */bin*.
dict:	Dictionaries for word processing programs.
games:	Various demonstration and instructional programs.
include:	Public C language *#include* files.
lib:	Archive libraries, including the text processing macros; also contains data files for various programs, such as *spell*(1) and *cron*(1M).
mail:	Mail directory.
man:	Source for the *UNIX/TS User's Manual*; see *man*(1).
mdec:	Hardware bootstrap loaders and programs.
news:	Place for all the various system news.
pub:	Handy public information, e.g., table of ASCII characters.
spool:	Spool directory for daemons.
src:	Source for commands, libraries, the system, etc.
tmp:	Directory for temporary files; should be cleaned at reboot.

A table of contents of this tape may be obtained by using the *cpio*(1) options *−itB*. Also, after installation, files and directories deemed useless by the local administrator may be easily removed. Alternately, only parts of the tape may be extracted using the pattern matching capabilities of *cpio*(1).

2.5 Initial Load of /usr

Mount a file system (device) as */usr*. The ultimate size and location of this file system on a device is an administrative decision; initially, the following procedure will suffice:

```
: "The 4 echo's will move the tape to file 5."
echo </dev/mt4
echo </dev/mt4
echo </dev/mt4
echo </dev/mt4
cd /
mkfs /dev/rrp1 35000 7 418
: If you have RP03 drives, the last argument above should be 200.
labelit /dev/rrp1 usr pkname
mount /dev/rp1 /usr
cd /usr
cpio −idmB </dev/rmt0
```

where *pkname* is the volume name of the pack (e.g., "p0001").

Since */usr* must be mounted when the system is in *multi-user* mode, the file */etc/rc* must be changed to include the command lines:

```
mount /dev/rp1 /usr
```

and

```
umount /dev/rp1
```

These lines must be inserted at the appropriate places in */etc/rc*, as indicated by comments in the prototype file. Next the file */etc/checklist* should be changed to include */dev/rrp1*. See also *fsck*(1M), *labelit*(1M), *mkfs*(1M), *mount*(1M).

2.6 Update of /usr

It is advisable that the system be running in *single-user* mode during the update phase. It is also wise to first make a copy of your */usr* file system for backup purposes. Next, mount the distribution tape on drive 0 and position it at file 5. The */usr* file system *must* also be mounted. The following procedure will perform the update:

```
cd /usr
cpio -idmB </dev/rmt0
```

3. CONFIGURATION PLANNING

3.1 UNIX/TS Configuration

The basic UNIX/TS operating system supplied supports only the console, a disk controller (disk drive 0), and a tape controller (tape drive 0). The actual configuration of your system must be described by you. All of the UNIX/TS operating system source code and object libraries are in */usr/src/uts*. All of the configuration information is kept in the directory */usr/src/uts/cf*. There are only two files that must be changed to reflect your system configuration, *low.s* and *conf.c;* the program *config*(1M) makes this task relatively simple.

Config requires a *system description file* and produces the two needed files. The first part of the system description file lists all of the hardware devices on your system. Next, various system information is listed. A brief explanation of this information follows. For more details of syntax and structure, see *config*(1M) and the associated *master*(5).

- *root*—Specifies the device where the *root* file system is to be found. The device must be a block device with read/write capability since this device will be mounted read/write as "/". Thus, a tape can not be mounted as the *root*, but can be mounted as some read-only file system. Normally, *root* is disk drive 0, section 0.
- *swap*—Specifies the device and blocks that will be used for *swapping*. *Swplo* is the first block number used and *nswap* indicates how many blocks, starting at *swplo*, to use. *Swplo* can not be zero. Typically, systems require approximately 2,000 blocks. Care must be taken that the swap area specified does not overlap any file system. For example, if section 0 is 8,000 blocks long, the *root* file system could occupy the first 6,000 blocks and *swap* the remaining 2,000 by specifying:

  ```
  root rp06 0
  swap rp06 0 6000 2000
  ```

- *pipe*—Specifies where pipes are to be allocated (must be a file system—the root file system is normally used).
- *dump*—Specifies the device to be used to dump memory after a system crash. Currently only the TU10 and TU16 tape drives are supported for this purpose.
- *buffers*—Specifies how many *system buffers* to allocate. In general, you will want as many buffers as possible without exceeding the system space limitations. Normally, *buffers* is in the range of 24—50. Each entry takes up 512 bytes *outside* system address space and 26 bytes *inside* the system.
- *sabufs*—Specifies how many *system addressable* buffers to allocate. One buffer is needed for every mounted file system. Certain I/O drivers need such buffers. Normally, *sabufs* is in the range 10—15. Each entry requires 540 bytes.
- *inodes*—Specifies how many *inode table* entries to allocate. Each entry represents a unique open inode. When the table overflows, the warning message "Inode table overflow" will be printed on the console. The table size should be increased if this happens regularly. The number of entries used depends on the number of active processes, texts, and mounts. Normally, *inodes* is in the range of 100—150. Each entry requires 74 bytes.
- *files*—Specifies how many *open-file table* entries to allocate. Each entry represents an open file. When the table overflows, the warning message "no file" will be printed on the console. The table size should be increased if this happens regularly. Normally, *files* is in the same range as the number of inodes. Each entry requires 8 bytes.
- *mounts*—Specifies how many *mount table* entries to allocate. Each entry represents a mounted file system. The *root* (/) will always be the first entry. When full, the *mount*(2) syscall will return the error EBUSY. Normally, *mounts* is in the range of 8-16. Each entry requires 10 bytes.
- *coremap*—Specifies how many entries to allocate to the *list of free memory*. Each entry represents a contiguous group of 64-byte blocks of free memory. When the list overflows,

due to excessive fragmentation, the system will undoubtedly crash in an unpredictable manner. The number of entries used depends on the number of processes active, their sizes, and the amount of memory available. Normally, *coremap* is in the range of 50-100. Each entry requires 4 bytes.

- *swapmap*—Specifies how many entries to allocate to the *list of free swap blocks*. Exactly like the coremap, except it represents free blocks in the swap area, in 512-byte units. Each entry requires 4 bytes.
- *calls*—Specifies how many *callout table* entries to allocate. Each entry represents a function to be invoked at a later time by the clock handler. The time unit is 1/60 of a second. The callout table is used by the terminal handlers to provide terminal delays and by various other I/O handlers. When the table overflows, the system will crash and print the panic message "Timeout table overflow" on the console. Normally, *calls* is in the range of 30-60. Each entry requires 6 bytes.
- *procs*—Specifies how many *process table* entries to allocate. Each entry represents an active process. The scheduler is always the first entry and *init*(8) is always the second entry. The number of entries depends on the number of terminal lines available and the number of processes spawned by each user. The average number of processes per user is in the range of 2-5. When full, the *fork*(2) syscall will return the error EAGAIN. Normally, *procs* is in the range of 50-200. Each entry requires 28 bytes.
- *texts*—Specifies how many *text table* entries to allocate. Each entry represents an active read-only text segment. Such programs are created by using the −i or −n option of the loader *ld*(1). When the table overflows, the warning message "out of text" is printed on the console. Normally, *texts* is in the range of 25-50. Each entry requires 12 bytes.
- *clists*—Specifies how many *character list buffers* to allocate. Each buffer contains up to six bytes. The buffers are dynamically linked together to form input and output queues for the terminal lines and various other slow-speed devices. The average number of buffers needed per terminal line is in the range of 5-10. When full, input characters from terminals will be lost and not echoed. Normally, *clists* is in the range of 100-300. Each entry requires 8 bytes.

3.2 UNIX/TS Generation

To generate a new UNIX/TS operating system, make sure that the directories under /usr/src/uts are up-to-date. Follow the procedure below:

```
cd /usr/src/uts/cf
ed dfile
a
    (information as described above)
.
w
q
make NAME=name CONFIG=dfile
```

Dfile is the name of the file containing the configuration information, and *name* is a character string of at most eight characters used to identify a specific system (such as "utsa0501"). The procedure will assemble *low.s*, compile *conf.c*, and load them together with the object libraries into a file called *name*.

The system has a finite address space, so that if table sizes or the number of device types are too large, various error messages will result and the above procedure will only create an *a.out* file. In particular, the maximum available data space is 49,152 bytes. The actual data space requested can be found by using *size*(1) on *a.out* and adding the *data* and *bss* segment sizes. One then reduces the specified values for the various system entries until it all fits. The amount of space in the *bss* segment used for each entry is indicated in Section 3.1 above.

When you are satisfied with the new system, you can test it by the following procedure:

```
cd /usr/src/uts
cp name /
cd /
rm /unix
ln /name /unix
sync
```

Then halt the processor and reboot *unix*. Note that this procedure results in two names for the operating system object, the generic */unix*, and the actual name, say */utsa0501*. An old system may be booted by referring to the actual name, but remember that many programs use the generic name */unix* to obtain the *name-list* of the system.

If the new system does not work, verify that the correct device addresses and interrupt vectors have been specified. If the *wrong* interrupt vector and the *correct* device address have been specified for a device, the operating system will print the error message "stray interrupt at XXX" when the device is accessed, where XXX is the correct interrupt vector. If the *wrong* device address is specified, the system will crash with a panic trap of type 0 (indicating a UNIBUS® timeout) when the device is accessed.

3.3 Special Files

A special file must be made for every device on your system. Normally, all special files are located in the directory */dev*. Initially, this directory will contain:

console	console terminal
error	see *err*(4)
mem, kmem, null	see *mem*(4)
tty	see *tty*(4)
rp[0−7], rrp[0−7]	disk drive 0, sections 0−7
mt0, rmt0	tape drive 0 (800 bpi)
mt4, rmt4	tape drive 0 (800 bpi, no rewind).

There are two types of special files, block and character. This is indicated by the character *b* or *c* in the listing produced by *ls*(1) with the "−l" flag.

In addition, each special file has a major device number and a minor device number. The major device number refers to the device type and is used as an index into either the *bdevsw* or *cdevsw* table in the configuration file *conf.c*. The minor device number refers to a particular unit of the device type and is used only by the driver for that type. For example, using the following sample portion of a configuration file, a block special file with major device number 1 and minor device number 0 would refer to the TU10 magtape, drive 0, while a character special file with major device number 1 and minor device number 4 would refer to the DH11 asynchronous multiplexor, line 4.

```
int     (*bdevsw[])()
{
/* 0*/  &hpopen,   &hpclose,   &hpstrategy,   &hptab,
/* 1*/  &tmopen,   &tmclose,   &tmstrategy,   &tmtab,
};
int     (*cdevsw[])()
{
/* 0*/  &klopen,   &klclose,   &klread,   &klwrite,   &klsgtty,
/* 1*/  &dhopen,   &dhclose,   &dhread,   &dhwrite,   &dhsgtty,
/* 2*/  &nulldev,  &nulldev,   &mmread,   &mmwrite,  &nodev,
/* 3*/  &nodev,    &nodev,     &nodev,    &nodev,    &nodev,
/* 4*/  &nodev,    &nodev,     &nodev,    &nodev,    &nodev,
/* 5*/  &nodev,    &nodev,     &nodev,    &nodev,    &nodev,
/* 6*/  &tmopen,   &tmclose,   &tmread,   &tmwrite,  &nodev,
/* 7*/  &hpopen,   &hpclose,   &hpread,   &hpwrite,  &nodev,
/* 8*/  &nodev,    &nodev,     &nodev,    &nodev,    &nodev,
/* 9*/  &nodev,    &nodev,     &nodev,    &nodev,    &nodev,
/*10*/  &nodev,    &nodev,     &nodev,    &nodev,    &nodev,
/*11*/  &nodev,    &nodev,     &nodev,    &nodev,    &nodev,
/*12*/  &nodev,    &nodev,     &nodev,    &nodev,    &nodev,
/*13*/  &syopen,   &nulldev,   &syread,   &sywrite,  &sysgtty,
};
```

The program *mknod*(1M) creates special files. For example, the following would create *part* of the initially-supplied */dev* directory:

```
cd /dev
mknod console c 0 0
mknod error c 20 0
mknod mem c 2 0; mknod kmem c 2 1; mknod null c 2 2
mknod tty c 13 0
mknod rp0 b 0 0; mknod rrp0 c 7 0
mknod mt0 b 1 0; mknod rmt0 c 6 0
mknod mt4 b 1 4; mknod rmt4 c 6 4
```

After the special files have been made, their access modes should be changed to appropriate values by *chmod*(1). For example:

```
cd /dev
chmod 622 console
chmod 444 error
chmod 644 mem kmem
chmod 666 null
chmod 666 tty
chmod 400 rp0 rrp0
chmod 666 mt0 rmt0
chmod 666 mt4 rmt4
```

Note that file names have no meaning to the *operating system* itself; only the major and minor device numbers are important. However, many *programs* expect that a particular file is a certain device. Thus, by convention, special files are named as follows:

block device	*conf.c*	*/dev*
RP03 disk	rp	rp*
RP04/5/6 disk	hp	rp*
RS03/4 fixed head disk	hs	rs*
TU10 tape	tm	mt*
TU16 tape	ht	mt*

character device	*conf.c*	*/dev*
DL11 asynch line	kl	tty•, console
DH11 asynch line mux	dh	tty•
DZ11 asynch line mux	dz	tty•
DN11 auto call unit	dn	dn•
DU11 synch line	du	du•
DQS11B synch line	dqs	rjei
KMC11 micro	kmc	kmc•
LP11 line printer	lp	lp•
RP03 disk	rp	rrp•
RP04/5/6 disk	hp	rrp•
RS03/4 fixed head disk	hs	rrs•
TU10 tape	tm	rmt•
TU16 tape	ht	rmt•
error	err	error
memory	mm	mem, kmem, null
terminal	sy	tty

For those devices with a */dev* name ending in "•", this character is replaced by a string of digits representing the *minor* device number. For example:

```
mknod /dev/mt1 b 1 1
mknod /dev/rp24 b 0 024
```

Note that for disks, an octal number scheme is maintained because each drive is split eight ways. Thus, */dev/rp24* refers to section 4 of physical drive 2. There is also a special file, */dev/swap*, that is used by the program *ps*(1). This file must reflect what device is used for swapping and must be readable. For example:

```
rm /dev/swap
mknod /dev/swap b 0 0
chmod 444 /dev/swap
```

3.4 File Systems

Each physical pack is split into eight logical sections. This split is defined in the operating system by a table with eight entries. Each table entry is two words long. The first specifies how many blocks are in the section, the second specifies the starting cylinder. See *hp*(4) (RP04/5/6) and *rp*(4) (RP03) for default cylinder/block assignments.

These values are described to the system in the header file */usr/include/sys/io.h* which may be changed by using the editor *ed*(1). After such a change, the system must be made again (see Section 3.2).

A file system starts at block 0 of a section of the disk and may be as large as the size of that section; if it is smaller than the size of a section, the remainder of that section is unused. Note that the sections themselves may overlap physical areas of the pack, but the file systems must *never* overlap.

The program *mkfs*(1M) is used to initialize a section of the disk to be a file system. Next, the program *labelit*(1M) is used to label the file system with its name and the name of the pack. Finally, the file system may be checked for consistency by using *fsck*(1M). The file system may then be mounted using *mount*(1M).

4. ADMINISTRATIVE FILES

4.1 /etc/motd

This file contains the *message-of-the-day.* It is printed by *login*(1) after every successful *login.*

4.2 /etc/rc

On the transition between init states, */etc/init* invokes */bin/sh* to run */etc/rc* (must have executable modes). So that */etc/rc* can properly handle the removal of temproary files and the mounting and unmounting of file systems, it is invoked with three arguments: new state, number of times this state has been entered, previous state. When the system is initially booted, */etc/rc* is invoked with arguments "1 0 0"; when state two(multi-user) is subsequently entered, the arguments are "2 0 1".

Daemons may be invoked either by */etc/rc* or by including lines for them in */etc/inittab*.

This file must be edited to suit local conditions; see *init*(8).

4.3 /etc/inittab

This file indicates to */etc/init* which processes to create in each init state. By convention, state 1 is *single-user* and state 2 is *multi-user*. For example, the following line creates the single-user environment:

```
1:co:c:/bin/-sh </dev/console >/dev/console 2>/dev/console
```

This indicates that for state 1 a process with the arbitrary unique identifier "co" should be created. The program invoked for this process should be the shell and when it exits it should be reinvoked ('c' flag).

To attach a *login* process to the console in the multi-user state, add the line:

```
2:co:c:/etc/getty console 4
```

and for line */dev/tty00* for use by 300/150/110 baud terminals, add the following line:

```
2:00:c:/etc/getty tty00 0 60
```

The arguments to getty are the device, speed table, and number of seconds to allow before hanging up the line.

Again, this file must be edited for local conditions. See *getty*(1M), *init*(8), *inittab*(5).

4.4 /etc/passwd

This file is used to describe each *user* to the system. You must add a new line for each new user. Each line has six fields separated by colons:

1. Login name:
   Normally 1-6 characters, first character alphabetic, rest alphanumeric, no upper-case.
2. Encrypted password:
   Initially null, filled in by *passwd*(1). The encrypted password contains 13 bytes, while the actual password is limited to a maximum of 8 bytes. The encrypted password may be followed by a comma and up to 4 more bytes of password age information.
3. User ID:
   A number between 0 and 65535; 0 indicates the super-user. These other IDs are reserved:

bin::2:	software administration;
sys::3:	system operation;
adm::4:	system administration;
uucp::5:	UNIX-UNIX file copy;
rje::68:	remote job entry administration;
games::196:	miscellaneous; never a real user.

4. Group ID:
   The default is group "1" (one).
5. Accounting information:
   This field is used by various accounting programs. It usually contains the user name,

department number, and account number.

6. Login directory:
   Full pathname (keep them reasonably short).

7. Program name:
   If null, */bin/sh* is invoked after successful *login*. If present, the named program is invoked in place of */bin/sh*.

For example:

```
ghh::38:1:6824-G.H.Hurtz(4357):/usr/ghh:
grk::44:1:6510-S.P.LeName(4466):/usr/grk:/bin/rsh
```

See also *passwd*(5), *login*(1), *passwd*(1).

4.5 /etc/group

This file is used to describe each *group* to the system. Each line has four fields separated by colons:

group name;
encrypted password;
numerical group ID;
list of all *login* names in the group, separated by commas.

See also *group*(5).

4.6 /etc/checklist

This file contains a list of default devices to be checked for consistency by the *fsck*(1M) program. The devices normally correspond to those mounted when the system is in *multi-user* mode. For example, a sample checklist would be:

```
/dev/rrp0
/dev/rrp1
```

4.7 /etc/shutdown

This file contains procedures to gracefully shut down the system in preparation for filesave or scheduled downtime.

4.8 /etc/filesave.?

This file contains the detailed procedures for the local filesave.

4.9 /usr/adm/pacct

This file contains the process accounting information. See *acct*(1M).

4.10 /usr/adm/wtmp

This file is the log of login processes.

5. REGENERATING SYSTEM SOFTWARE

System source is issued under the directory */usr/src*. The sub-directories are named *cmd* (commands), *lib* (libraries), *uts* (the operating system), *head* (header files), and *util* (utilities). See *mk*(8) for details on how to remake system software.