Unix/v7m Release 2.1 - Software Description

Fred Canter

Digital Equipment Corporation
Continential Boulevard
Merrimack, New Hampshire Ø3Ø54

UNIX* version seven modified, `Unix/v7m' is a full source distribution of Unix version seven plus the enhancements developed by the Unix Systems Engineering group at Digital. The latest Western Electric addenda tape, dated 12/2/8Ø is also included on the Unix/v7m distribution tape, see `/sys/addenda/README' for more details about the addenda. The documents `Setting up Unix', `Regenerating System Software', and the accompanying manual pages supersede the equivalent standard Unix version documents.

*UNIX is a Trademark of Bell Laboratories.

September 22, 1981

Distribution Format

The distribution tape is available at two densities, 800 BPI for use with the TU10/TE10 and TU16/TE16 tape drives and 1600 BPI for use with the TS11 and TU16/TE16 tape drives. The 1600 BPI tape can be used for the TU16/TE16 only if the TM03 tape controller (WITH AUTO DENSITY SELECT ENABLED) is used, the 1600 BPI tape cannot be used with the TM02 tape controller.

The distribution tape consists of some preliminary bootstrapping programs followed by a root file system image in dump/restor format, the system sources in tar format, and a tar formatted user file system with the remainder of the sources.

The system as distributed contains binary images of the system and all the user level programs, along with source and manual sections for them-about 2100 files altogether. The binary images, along with other things needed to flesh out the file system enough so UNIX will run, are to be put on one file system called the `root file system'. The file system size required is 9600 blocks. The second file system contains the system sources, needed to rebuild Unix and requires about 5000 512-byte blocks. The third file system has the remainder of the sources and all of the documentation, it requires 20,000 blocks of storage.

Unix/v7m Versions

The Unix/v7m tape includes three versions of the Unix operating system; `unix_id', the separated Instruction and Data space version for use with the PDP11/44, PDP11/45, PDP11/55, and PDP11/70 processors; `unix_ov', the overlay text kernel used with the non-separated Instruction and Data space processors, PDP11/23, PDP11/24, PDP11/34, PDP11/40, and PDP11/60; `unix_i', the preconfigured unix monitors used only for the initial loading of Unix from the distribution tape.

The separated I and D space version takes full advantage of the features in Unix version seven and can support up to 32 users, depending on the processor type and system configuration. The overlay text kernel is designed to support about 10 users, more or less depending on the processor and system configuration. The overlay version limits the size of user processes to 64 kb, due to the lack of separated I and D space in the processor.

The overlay unix kernel used in this distribution was supplied by Mr. Bill Shannon, thanks to Bill for a job well done !

Hardware Requirements

Unix/v7m, with the exceptions listed below, can be used on any system that is configured with a processor, one disk, and a tape from the following equipment list:

CPU	DISK	TAPE
PDP11/23	RL01/2	TU10
PDP11/24	RK06/7	TE10
PDP11/34	RM02/3	TS11
PDP11/40	RP03	TU16
PDP11/44	RP04/5/6	TE16
PDP11/45		
PDP11/55		
PDP11/60		
PDP11/70		

OPTIONAL EQUIPMENT

DISKS	TAPES	PRINTERS	COMMUNICATIONS	
ML11	TS03	LP11	DH11	
RS03/4	TU56		DM11	(DH modem control)
RX02			DZ11	
RK05			DN11	
RF11			DU11	
			DL11	
			DC11	

1. The processor must be equiped with a KW11-L line clock or a KW11-P programmable clock.

2. The `practical' minimum memory size for Unix/v7m on all processors is 256 K bytes. A memory size of 192 K bytes is acceptable, however the number of users will be limited. Unix/v7m is NOT guaranteed to work with less than 192 K bytes of main memory. In fact, on a PDP11/23 with 128 k bytes of memory it did not function properly. For the most part it worked, but some important things, such as the floating point simulator (cc -f) and commands that used floation point, such as iostat(1), did not work. Also, the secondary bootstrap program `boot' will not function with less than 192 K bytes.

3. All of the FP11 type floating point processors are supported by Unix/v7m. The PDP11/40 Float Instruction Set is not supported by Unix. A floating point simulator is available for processors which are not equipted with floating point hardware. However this code is very slow and if any serious floating point work is to be done the floating point hardware is required.

4. Four RL01 disk packs are required to contain this distribution, however, Unix/v7m can be operated on as few as two RL01 drives, three are recommended.

5. With the RL02 and RK06 disks two drives are required.

6. The RK05 disk is not large enough to be the system (root) device, it is supported as a user device only.

7. The RMØ3 disk can only be used with the PDP11/7Ø, all other processors must use the RMØ2.

8. The RXØ2 can be used in single and double density modes, the RXØ1 is not supported.

9. The ML11 solid state disk may be connected to the same massbus with the system disk or on a separate RH11 or RH7Ø massbus disk controller.

1Ø. The RSØ3/4 and the ML11 are mutually exclusive.

11. The TSØ3 tape is supported as a user device only, because it cannot accept reels of tape that are greater than 6ØØ feet in length.

12. For the TS11 a single drive only is supported. A TS11 and a TM11 (TU1Ø/TE1Ø) may be configured on the same system, with certain restrictions. The TM11 controller must be at CSR address 17252Ø and vector 224 and the TS11 must be at address 172550 and vector 26Ø. The TS11 can be booted as `MS6', assuming that the system is equiped with a M9312 ROM bootstrap using the console emulator. Unix/v7m will automatically adapt to this configuration, the TM11 will be accessed by `mtØ' and the TS11 by `mt1'. The `Setting up Unix' document has instructions , flaged by `tstm', which pertain to this configuration. If the TS11 is the only tape on the system it must be at CSR address 172520 and vector 224.

13. An additional RH massbus disk controller, with a combination of RMØ2/3, and/or RPØ4/5/6 disks attached, is also supported. This configuration is referred to in `Setting up Unix' as the `hm' disk. There is no hardware boot ROM available to bootstrap from a disk on the second RH controller, however, it can be booted by calling into memory a copy of `boot' from another disk or the distribution tape.

PDP11/23 Anomalies

There are certain unexplained error conditions which cause the PDP11/23 to enter a state where it cannot be rebooted without powering the system off and then back on again. When the PDP11/23 is booted it halts instead of starting the primary bootstrap program, type `Øg' to start the primary bootstrap.

A Word About Disk Media

Unix expects the disk media on which it lives to be perfect, i.e., no bad blocks. The Unix file system places things (super block, i list, free list) in fixed positions on the disk, there is no strategy for mapping around bad blocks. The Unix device drivers for disks with error correcton capability (hp & hk) implement ECC in both block and raw I/O modes. All other disk drivers depend on retries as their only means of error recovery. A bad disk block is one that contains an error which cannot be corrected via ECC, for disks with ECC, or any error which cannot be recovered via retry, for disks without ECC. Retry and ECC recoverable errors are "soft" errors. Certain disk packs (RMØ2/3 and RKØ6/7) have the bad blocks flagged in the sector header. The disk hardware will not attempt to write or read these blocks, but instead will report a bad sector error (BSE).

To qualify a disk pack for use with Unix, First, obtain a list of the locations of all bad blocks and soft errors on the disk pack. The DEC format and verify diagnostic should provide this information when the disk pack is formatted. Compare the error location list to the file system layout for the type of disk to be used, the following guide lines should be used:

1. The root file system must be free of bad blocks and although soft errors can be corrected they should be avoided if possible. Soft errors can not be tolerated in any disk block that is allocated to the file "/unix" or any other bootable file, because the standalone bootstrap does not implement ECC.

2. The swap area must be free of bad blocks. Soft errors should be avoided if possible, because the time required to correct them will degrade system performance.

3. The user file systems must be free of bad blocks, soft errors are generally not a problem in the user areas. It is sometimes possible to use a disk pack with bad blocks in the user file systems by rearranging the disk partitions to avoid the bad blocks.

Release notes

The following notes give a very brief description of the differences between this distribution and Unix/v7m release one. The release notes are attention getters only, refer to the documentation for more detailed information about the changes and new features.

1. The `#' prompt has been added to the block zero disk bootstraps, see boot(8).

2. The overlay Unix kernel must be booted with the two stage bootstrap, see boot(8).

3. The RKØ6 and RKØ7 block zero disk boot programs have been combined into `hkuboot'.

4. The block zero disk bootstraps would halt if only a return was typed, they will now return to the `#' prompt.

5. The old (unix_i) I space only Unix systems may still be booted directly, without using the two stage bootstrap.

6. The `-p' option has been added to the ls(1) command, see ls(1).

7. Logins may now be disabled/enabled at the console terminal, see logins(1m).

8. There is now system shutdown command, see shutdown(8).

9. The iostat(1m) command has been completely rewritten, see iostat(1m).

1Ø. The fsck(1m) command was added, see fsck(1m).

11. The ps(1) and pstat(1m) commands no longer rely on the param.h file for system parameter information, they read it from the unix kernel directly, using the namelist and /dev/mem facilities.

12. Several other commands have been changed, see the documentation for the details.

13. The file `/sys/addenda/README' is must reading !

14. New processors and disks are supported, see hardware requirements above.

15. Local terminals are now supported , see ttys(5).

16. The init(8) program has been modified to reread the ttys file, see init(8).

17. Many floating point changes have been made, see the documentation for more detail.

18. A TS11 and A TM11 magtape on the same system is now supported.

19. Several bugs in the core dump code were fixed. The new core dump starting address is now 1000 instead of 44.

20. The RK06/7 disk driver has been completely rewritten and the R101/2 driver has been improved.

21. The RM02/3, ML11, and RP04/5/6 disks on the second RH11 or RH70 are now supported, see hm(4).

22. The HP and HK disk drivers now have full ECC in block and raw I/O modes.

23. Memory parity error handling has been improved.

24. Stray vector handling has been improved.

25. Disk free list spacing has been optimized for disk rotation.

26. The `mkconf' program has been extensively modified, see mkconf(1m).

Setting Up Unix - Seventh Edition

Charles B. Haley
Dennis M. Ritchie

Bell Laboratories
Murray Hill, New Jersey 07974

This document has been updated to include additional processor and peripheral device support , and other enhancements incorporated in UNIX* version seven by the Unix Systems Engineering group at DIGITAL. The addenda tape (dated 12/2/80) supplied with the standard Unix version seven distribution has been included on this tape in the directory /sys/addenda. Most of the addenda has already been installed on this distribution, for more detail about the addenda see /sys/addenda/README.

Unix/v7m Release 2.1

Fred Canter
Bill Shannon
Jerry Brenner
Armando Stettner

Digital Equipment Corporation
Continential Boulevard
Merrimack, New Hampshire 03054

*UNIX is a Trademark of Bell Laboratories.

September 23, 1981

If you are set up to do it, it might be a good idea immediately to make a copy of the tape to guard against disaster. The tape is 9-track 800 BPI or 1600 BPI and contains some 512-byte records followed by many 10240-byte records. There are interspersed tapemarks.

The system as distributed contains binary images of the system and all the user level programs, along with source and manual sections for them-about 2100 files altogether. The binary images, along with other things needed to flesh out the file system enough so UNIX will run, are to be put on one file system called the `root file system'. The file system size required is 9600 blocks. The second file system contains the system sources, needed to rebuild Unix and requires about 5000 512-byte blocks. The third file system has the remainder of the sources and all of the documentation, it requires 20,000 blocks of storage.

Making a Disk From Tape

Perform the following bootstrap procedure to obtain a disk with a root file system on it.

1. Mount the magtape on drive 0 at load point.

2. Mount a formatted disk pack on drive 0.
   In some cases a disk pack must also be mounted on drive 1.

3. Use the DEC hardware boot ROM or other means to load block 0 or 1 at 800 or 1600 BPI into location 0 and transfer to 0.

4. The tape should move and the CPU loop.

5. The console should type

```
Boot
:
```

   Copy the magtape to disk by the following procedure. The machine's printouts are shown in " ", explanatory comments are within ( ). Terminate each line you type by carriage return or line-feed. There are three classes of tape drives: the name `tm' is used for the TU10 or TE10, `ts' is used for the TS11, and `ht' is used for the TU16 or TE16. There are also four classes of disks: `rp' is used for the RP03, `hp' is used for the RM02/3 and the RP04/5/6, `hk' is used for the RK06/7 , and `rl' is used for the RL02.

If you should make a mistake while typing, the character '#' erases the last character typed up to the beginning of the line, and the character '@' erases the entire line typed. Some consoles cannot print lower case letters, adjust the instructions accordingly.

```
(bring in the program mkfs)
":" tm(0,3)      (`tm(0,3)' for TU10/TE10 800 BPI)
                 (`ht(0,3)' for TU16/TE16 800 BPI)
                 (`ht(4,3)' for TU16/TE16 1600 BPI)
                 (`ts(0,3)' for TS11 1600 BPI)
                 (`ts(6,3)' for TS11 1600 BPI `tstm')

"file system size:" 9600        (9600 for RP03)
                          (9600 for RM02/3, RP04/5/6)
                          (9600 for RK06/7)
                          (7240 for RL01)
                          (16480 for RL02)

(see Table 1, in Disk Layout below, for values of `m' and `n')
"interleave factor:" m

"blocks per cylinder:" n

"file system:" rp(0,0)   (`rp(0,0)' for RP03)
                         (`hp(0,0)' for RM02/3, RP04/5/6)
                         (`hm(0,0)' for hp on second RH)
                         (`hk(0,0)' for RK06/7)
                         (`rl(0,0)' for RL01/2)
"isize = XX"
"m/n = XX"
(after a while)
"exit called"
"Boot"
":"
```

The above step makes an empty file system.

6. The next thing to do is to restore the data onto the new empty file system. To do this you respond to the `:' printed in the last step with

```
(bring in the program restor)
":" tm(0,4)      (`tm(0,4)' for TU10/TE10 800 BPI)
                 (`ht(0,4)' for TU16/TE16 800 BPI)
                 (`ht(4,4)' for TU16/TE16 1600 BPI)
                 (`ts(0,4)' for TS11 1600 BPI)
                 (`ts(6,4)' for TS11 1600 BPI `tstm')

"tape?" tm(0,5)  (`tm(0,5)' for TU10/TE10 800 BPI)
                 (`ht(0,5)' for TU16/TE16 800 BPI)
                 (`ht(4,5)' for TU16/TE16 1600 BPI)
                 (`ts(0,5)' for TS11 1600 BPI)
                 (`ts(6,5)' for TS11 1600 BPI `tstm')

"disk?" rp(0,0)  (`rp(0,0)' for RP03)
                 (`hp(0,0)' for RM02/3, RP04/5/6)
                 (`hm(0,0)' for hp on second RH)
                 (`hk(0,0)' for RK06/7)
                 (`rl(0,0)' for RL01/2)
"Last chance before scribbling on disk." (you type return)
(the tape moves, perhaps 5-10 minutes pass)
"end of tape"
"Boot"
":"
```

You now have a UNIX root file system.

Booting UNIX

You probably have the bootstrap running, left over from the last step above; if not, repeat the boot process (step 3) again. The root file system contains 18 preconfigured unix operating systems. These unix systems are named `xyunix', where `x' is the disk name and `y' is the tape name. Boot the unix monitor which most closely matches your configuration by responding to the ":" with one of the following:

```
hp(0,0)hphtunix          (for RP04/5/6 or RM02/3,)
hp(0,0)hptmunix          (on first RH controller)
hp(0,0)hptsunix
hm(0,0)hmhtunix          (for RP04/5/6 or RM02/3,)
hm(0,0)hmtmunix          (on second RH controller)
hm(0,0)hmtsunix
rp(0,0)rphtunix          (for RP03)
rp(0,0)rptmunix
rp(0,0)rptsunix
hk(0,0)hkhtunix          (for RK06/7)
hk(0,0)hktmunix
hk(0,0)hktsunix
rl(0,0)rl01htunix        (for RL01)
rl(0,0)rl01tmunix
rl(0,0)rl01tsunix
rl(0,0)rl02htunix        (for RL02)
rl(0,0)rl02tmunix
rl(0,0)rl02tsunix
```

The machine should type the following:

```
unix/v7m 2.1

mem = xxx
#
```

The mem message gives the memory available to user programs in bytes.

After booting unix, the current date and time (date (1)) must be set as follows:

```
# date yymmddhhmm
```

i.e.

```
# date 8108231300
```

UNIX is now running, and the `UNIX Programmer's manual' applies; references below of the form X(Y) mean the subsection named X in section Y of the manual. If your processor is one of the non-separate I & D space CPUs mentioned above, you must use the alternate version of several system commands and the C object code optimizer. If required the alternate commands should be installed as follows:

```
cd /bin
cat makefile
make cmd40      (non-separate I & D CPU)
or
make cmd70      (separate I & D CPU)
cd /
sync
```

The system is distributed with the `cmd70' commands installed. The following commands will be replaced:

```
adb
dcheck
dump
dumpdir
icheck
ncheck
restor
tar
```

The above list represents the commands required to set up and maintain unix version seven on the smaller processors. Most of the remaining commands will function without being recompiled, however there will be certain commands that will need to be recompiled and others that are too large to operate on non-separate I & D space CPUs.

If your system disk is the RL01 or RL02, you must select alternate `/etc/rc' and `/etc/checklist' files as follows:

```
cp /etc/rc_rl01 /etc/rc
cp /etc/cl_rl01 /etc/checklist
or
cp /etc/rc_rl02 /etc/rc
cp /etc/cl_rl02 /etc/checklist
```

The `#' is the prompt from the Shell, and indicates you are the super-user. The user name of the super-user is `root' if you should find yourself in multi-user mode and need to log in; the login is `root', there is no password.

To simplify your life later, copy the appropriate version of the system as specified above to plain `unix.' For example, use cp (1) as follows if you have an RPØ3 disk and a TU16/TE16 tape:

```
cp rphtunix unix
```

In the future, when you reboot, you can type just

```
hp(Ø,Ø)unix
```

to the `:' prompt. (Choose appropriately among `hp', `rp', `hk', `rl' , `ht', `tm', `ts' according to your configuration). After selecting and copying the correct version of unix, all of the remaining preconfigured unix systems should be removed in order to free up space in the root file system. The appropriate preconfigured unix system should be retained so that in case of disaster there will always be a bootable unix system in the root.

You now need to make some special file entries in the dev directory. These specify what sort of disk you are running on, what sort of tape drive you have, and where the file systems are. For simplicity, this recipe creates fixed device names. These names will be used below, and some of them are built into various programs, so they are most convenient. However, the names do not always represent the actual major and minor device in the manner suggested in section 4 of the Programmer's Manual. For example, `rp3' will be used for the name of the file system on which the user file system is put, even though it might be on an RPØ6 and is not necessarily logical device 3. Also, this sequence will put the user file system on the same disk drive as the root, which is not the best place if you have more than one drive. Thus the prescription below should be taken only as one example of where to put things. See also the section on `Disk layout' below.

In any event, change to the dev directory (cd(1)) and, if you like, examine and perhaps change the makefile there (make (1)).

```
cd /dev
cat makefile
```

Then, use one of

```
make rpØ3
make rpØ4
make rpØ5
make rpØ6
make rmØ2
make rmØ3
make rpØ4_2
make rpØ5_2
make rpØ6_2
make rmØ2_2
make rmØ3_2
make rkØ6
make rkØ7
make rlØ1
make rlØ2
```

depending on which disk you have. Then, use one of

```
make tm
make ht
make ts
make tstm
```

depending on which tape you have. The `_2' lines are for the hp disks on the second RH11 or RH7Ø controller. The `tstm' line is for the special case of having a TS11 and a TM11 on the same system.

The file `rpØ' refers to the root file system; `swap' to the swap-space file system; `rp2' to the system source file system; 'rp3' to the user file system. The devices `rrpØ', 'rrp2', and `rrp3' are the `raw' versions of the disks. Also, `mtØ' is tape drive Ø, at 8ØØ BPI; `rmtØ' is the raw tape, on which large records can be read and written; `nrmtØ' is raw tape with the quirk that it does not rewind on close, which is a subterfuge that permits multi-file tapes to be handled. `mt1', `rmt1', and `nrmt1' also refer to tape drive Ø, but at 16ØØ BPI.

Before anything further is done the bootstrap block on the disk (block 0) should be filled in. This is done using one of the following commands:

```
(RP03)
dd if=/mdec/rpuboot of=/dev/rp0 count=1

(RM02/3, RP04/5/6)
dd if=/mdec/hpuboot of=/dev/rp0 count=1

(RK06/7)
dd if=/mdec/hkuboot of=/dev/rp0 count=1

(RL01/2)
dd if=/mdec/rluboot of=/dev/rp0 count=1
```

Now the DEC disk bootstraps are usable. See Boot Procedures(8) for further information.

The system sources should now be extracted from the distribution tape by following the procedure shown below, again the comments are enclosed in ( ). The number following the "mkfs" command is the size of the file system, which differs for each type of disk. The numbers `m' and `n', following the size , are the file system interleave factors, see `Disk Layout' table 1 to obtain the optimum values for your system.

```
cd /
        (make an empty file system)
/etc/mkfs /dev/rrp2 ##### m n   (skip this if using RL02)
                    (07400 on RP03)
                    (08000 on RM02/3)
                    (08778 on RP04/5/6)
                    (08514 on RK06/7)
                    (10240 on RL01)
/etc/mount /dev/rp2 /sys        (skip this if using RL02)
cd /sys
        (skip 6 files on the tape)
dd if=/dev/nrmt0 of=/dev/null bs=20b files=6    ( 800 BPI)
dd if=/dev/nrmt1 of=/dev/null bs=20b files=6    (1600 BPI)
        (extract the source files)
tar40 xbf 20 /dev/rmt0  ( 800 BPI)
tar40 xbf 20 /dev/rmt1  (1600 BPI)
cd /
sync
/etc/umount /dev/rp2    (skip this step if using RL02)
```

The next thing to do is to extract the rest of the data from the tape. This user file system is too large for a single RL01 disk, it must be split between two RL01 disk packs. There is no set procedure for accomplishing the split, one suggestion would be to load the source files on one pack, the documentation on the second pack and then fill out each pack with the remainder of the files. Comments are enclosed in ( ); don't type these. The number in the first command is the size of the file system; it differs between RP03, RM02/3, RP04/5, RP06 , RK06/7, and RL01/2. Again see `Disk Layout' table 1 for the values of `m' and `n'.

```
        (make an empty file system)
/etc/mkfs /dev/rrp3 ###### m n
                    (055000 on RP03)
                    (105120 on RM02/3)
                    (144210 on RP04/5)
                    (313082 on RP06)
                    (010240 on RL01)
                    (020480 on RL02)
                    (027060 on RK06)
                    (026598 on RK07)
(The above command takes about 2-3 minutes on an RP03)
/etc/mount /dev/rp3 /usr
cd /usr
        (skip 7 files on the tape)
dd if=/dev/nrmt0 of=/dev/null bs=20b files=7     ( 800 BPI)
dd if=/dev/nrmt1 of=/dev/null bs=20b files=7     (1600 BPI)
        (extract the files)
tar40 xbf 20 /dev/rmt0  ( 800 BPI)
tar40 xbf 20 /dev/rmt1  (1600 BPI)
(This takes a while, time for a break !)
cd /
sync
/etc/umount /dev/rp3
```

All of the data on the tape has been extracted.

You may at this point mount the source file systems (mount(1)). To do this type the following:

```
/etc/mount /dev/rp2 /sys (skip this step if using RL02)
/etc/mount /dev/rp3 /usr
```

The source and manual pages are now available in subdirectories of /usr and /sys. The above mount commands are only needed if you intend to play around with source on a single user system, which is not necessary. The file systems are mounted automatically when multi-user mode is entered, by a command in the file /etc/rc. (See `Disk Layout' below). The system should now be placed into multi-user mode by typing the control and d keys simultaneously (control d), some text will be printed followed by the "login:" prompt.

Before UNIX is turned up completely, a few configuration dependent exercises must be performed. At this point, it would be wise to read all of the manuals (especially `Regenerating System Software') and to augment this reading with hand to hand combat. DO NOT !, proceed past this point until you have read the section on regenerating UNIX in `Regenerating System Software'. The `Setting Up Unix' and the `Regenerating System Software' documents are located in /usr/doc and can be printed by using one of the following commands:

```
nroff -mf setup
nroff -mf regen
```

Reconfiguration

The general information given in this section is intended to document the areas of the unix software which require modification in order to tailor unix to the specific system configuration. The suggested procedure is to read this section and then follow the step by step instructions in the `UNIX' section of `Regenerating system Software'.

The first step is to select the type of unix system most appropriate to your CPU, there are three; unix_id for the separate I & D space CPUs (PDP11/44, PDP11/45, PDP11/55, & PDP11/70); unix_i for the non-separate I & D space CPUs (PDP11/23, PDP11/24, PDP11/34, PDP11/40, & PDP11/60); unix_ov the overlay text kernel for the non-separate I & D space CPUs. The size limitations of the unix_i kernel makes its use as an actual unix system inappropriate, its only purpose in life is to initially load unix from the distribution tape. The overlay text unix kernel should be used as the multi-user unix system on the non-separate I & D space machines.

The UNIX system running is configured to run on a non-separate I & D space CPU with the given disk and tape, a console, and no other device. This is certainly not the correct configuration. You will have to correct the configuration table to reflect the true state of your machine.

It is wise at this point to know how to recompile the system. Respond to the "login:" prompt with "sys" followed by a return, there is no password. This will log you into the system source account with a current directory of /sys. Print (cat(1)) the file /sys/conf/makefile. This file is input to the program `make(1)' which if invoked with `make all' will recompile all of the system source and install it in the correct libraries. The libraries supplied with this distribution tape are up to date and need not be recompiled unless changes are made to the system source code. See `Regenerating System Software' for instructions on recompiling individual source modules and installing them in the correct libraries.

The file /sys/h/param.h contains the parameters which determine the size of the various data spaces within unix. There are two versions of this file, /sys/h/param_ov.h for the PDP11/23, PDP11/24, PDP11/34, PDP11/40, and PDP11/60, and /sys/h/param_id.h for the PDP11/44, PDP11/45, and PDP11/70. Prior to recompiling any of the system source code the correct parameter file must be copied to /sys/h/param.h and to /usr/include/sys/param.h. This is done automatically by the makefiles and shell procedues which are provided for making unix.

The program mkconf(1) prepares files that describe a given configuration (See mkconf(1)). In the /sys/conf directory, the 18 files xyconf were input to mkconf to produce the 18 versions of the system xyunix, "x" is the disk type (rp, hp, hm, hk, rl01, or rl02) and "y" is the tape (ht, tm, or ts). Pick the appropriate one, copy it to unixconf, and edit unixconf to add lines describing your own configuration. (Remember the console typewriter is automatically included; don't count it in the kl specification.) Then run mkconf; it will generate the files l.s (trap vectors) c.c (configuration table), and mch0.s. Take a careful look at l.s to make sure that all the devices that you have are assembled in the correct interrupt vectors. If your configuration is non-standard, you will have to modify l.s to fit your configuration.

There are certain magic numbers and configuration parameters imbedded in various device drivers that you may want to change. The device addresses of each device are defined in each driver. In case you have any non-standard device addresses, just change the address and recompile. (The device drivers are in the directory /sys/dev.)

The DZ11 driver is configured for two DZ11 (8 lines).

The DC11 driver is set to run 4 lines. This can be changed in dc.c.

The DH11 driver is set to handle 1 DH11 with a full complement of 16 lines. If you have less, or more, you may want to edit dh.c.

The DN11 driver will handle 4 DN's. Edit dn.c.

The DU11 driver can only handle a single DU. This cannot be easily changed.

The KL/DL driver is set up to run a single DL11-A, -B, or -C (the console) and no DL11-E's. To change this, edit kl.c to have NKL11 reflect the total number of DL11-ABC's and NDL11 to reflect the number of DL11-E's. So far as the driver is concerned, the difference between the devices is their address.

You should edit of the disk and tape drivers (rl.c, rf.c, rk.c, rp.c, tm.c, tc.c, ts.c, hk.c, hp.c, ht.c) to reflect the number of disk and tape drives in your configuration. The big disk drivers (hk.c, rp.c, and hp.c) have partition tables in them which you may want to experiment with.

After all the corrections have been made, use `make(1)' to recompile the system (or recompile individually if you wish: use the makefile as a guide). If you compiled individually, say `make unix??' in the directory /sys/conf , the ?? is the CPU type, i.e., 23, 24, 34, 40, 60, 44, 45, 55, 70. The final object file will be named unix_ov for the PDP11/23, PDP11/24, PDP11/34, PDP11/40, and PDP11/60 or unix_id for the PDP11/44, PDP11/45, PDP11/55, and PDP11/70. This file should be moved to the root, and then booted to try it out. It is best to name it /nunix so as not to destroy the working system until you're sure it does work. See Boot Procedures(8) for a discussion of booting. Note: before taking the system down, always (!!) perform a sync(1m) to force delayed output to the disk.

Special Files

Next you must put in special files for the new devices in the directory /dev using mknod(1). Print the configuration file c.c created above. This is the major device switch of each device class (block and character). There is one line for each device configured in your system and a null line for place holding for those devices not configured. The essential block special files were installed above; for any new devices, the major device number is selected by counting the line number (from zero) of the device's entry in the block configuration table. Thus the first entry in the table bdevsw would be major device zero. This number is also printed in the table along the right margin.

The minor device is the drive number, unit number or partition as described under each device in section 4 of the manual. For tapes where the unit is dial selectable, a special file may be made for each possible selection. You can also add entries for other disk drives.

In reality, device names are arbitrary. It is usually convenient to have a system for deriving names, but it doesn't have to be the one presented above.

Some further notes on minor device numbers. The hp and hk drivers use the 0100 bit of the minor device number to indicate whether or not to interleave a file system across more than one physical device. See hp(4) and hk(4) for more detail. The tm, ts, and ht drivers use the 0200 bit to indicate whether or not to rewind the tape when it is closed. The 0100 bit indicates the density of the tape, set for 800 BPI and cleared for 1600 BPI. By convention, tape special files with the 0200 bit on have an `n' prepended to their name, as in /dev/nmt0 or /dev/nrmtl. Again, see tm(4), ts(4) or ht(4).

The naming of character devices is similar to block devices. Here the names are even more arbitrary except that devices meant to be used for teletype access should (to avoid confusion, no other reason) be named /dev/ttyX, where X is some string (as in `00' or `library'). The files console, mem, kmem, and null are already correctly configured. The makefile in /dev can be used to make the special files for the DZ11 and DH11 communications multiplexers as follows:

```
make tty_rm     (removes all tty special files)
make dz#        (# = dz unit number)
make dh#        (# = dh unit number)
```

Each dz11 unit handles 8 lines, and each dh11 unit 16 lines. Prior to making the dz or dh special files the "tty_rm" make command should be executed in order to remove an existing tty special files, the files "/dev/console" and "/dev/tty" will not be removed.

The disk and magtape drivers provide a `raw' interface to the device which provides direct transmission between the user's core and the device and allows reading or writing large records. The raw device counts as a character device, and should have the name of the corresponding standard block special file with `r' prepended. (The `n' for no rewind tapes violates this rule.) Thus the raw magtape files would be called /dev/rmtX. These special files should be made.

When all the special files have been created, care should be taken to change the access modes (chmod(1)) on these files to appropriate values (probably 600 or 644).

Floating Point

UNIX only supports (and really expects to have) the FP11 type floating point unit. The PDP11/40 Floating Instruction Set is NOT supported by unix. For machines without the FP11 hardware, there is a user subroutine available that will catch illegal instruction traps and interpret floating point operations. This allows programs with floating point constants to be compiled and executed, on machines without the FP11. To compile floating point programs use the `-f' flag to cc(1). This flag ensures that the floating point interpreter is loaded with the program and that the floating point version of `cc' is used. The system as delivered has this code included in only the iostat (1) and ac (1) commands and adb (1), although the operating system adapts automatically to the presence or absence of the FP11. If a program which has the floating point interpreter included is executed on a CPU with the FP11, the interpreter code is ignored and the floating point hardware is used instead. The floating point interpreter code may be removed by recompiling the program without the -f option. The floating point simulator software is extremely slow, if any serious floating point work is to be done the floating point hardware is really required.

The changes described in the document "Unix Problems With Floating-Point Processors" by Bob Campbell, Ed Gould, Vance Vaughan, and Jim Reeds of the University of California at Berkeley, have been installed in this distribution.

Time Conversion

If your machine is not in the Eastern time zone, you must edit (ed(1)) the file /sys/h/param.h to reflect your local time. The manifest `TIMEZONE' should be changed to reflect the time difference between local time and GMT in minutes. For EST, this is 5*60; for PST it would be 8*60. Finally, there is a `DSTFLAG' manifest; when it is 1 it causes the time to shift to Daylight Savings automatically between the last Sundays in April and October (or other algorithms in 1974 and 1975). Normally this will not have to be reset. When the needed changes are done, recompile and load the system using make(1) and install it. (As a general rule, when a system header file is changed, the entire system should be recompiled. As it happens, the only uses of these flags are in /sys/sys/sys4.c, so if this is all that was changed it alone needs to be recompiled.)

You may also want to look at timezone(3) (/usr/src/libc/gen/timezone.c) to see if the name of your timezone is in its internal table. If needed, edit the changes in. After timezone.c has been edited it should be compiled and installed in its library. (See /usr/src/libc/(mklib and compall)) Then you should (at your leisure) recompile and reinstall all programs that use it (such as date(1)).

Disk Layout

The following table lists the file system interleave factors (m and n) to be used with mkfs (1m) and fsck -s (1m) to achieve optimum free list spacing when creating or salvaging file systems. These values were derived through experimentation, calculation and intuition, and although they may not be optimal in all cases they are certainly better that the default free list spacing assumed by mkfs and fsck.

TABLE 1

File System Interleave Factors

CPU TYPE	RLØ1/2 m/n	RKØ6/7 m/n	RPØ3 m/n	RMØ2 m/n	RMØ3 m/n	RPØ4/5/6 m/n
11/23	13/2Ø	X	X	X	X	X
11/24	12/2Ø	14/66	X	2Ø/16Ø	X	2Ø/418
11/34	1Ø/2Ø	11/66	6/2ØØ	15/16Ø	X	15/418
11/4Ø	11/2Ø	12/66	6/2ØØ	16/16Ø	X	16/418
11/44	7/2Ø	8/66	4/2ØØ	11/16Ø	X	11/418
11/45	9/2Ø	1Ø/66	5/2ØØ	14/16Ø	X	14/418
11/45z	11/2Ø	12/66	X	17/16Ø	X	17/418
11/55	9/2Ø	1Ø/66	5/2ØØ	14/16Ø	X	14/418
11/6Ø	9/2Ø	1Ø/66	5/2ØØ	14/16Ø	X	14/418
11/7Ø	6/2Ø	6/66	X	9/16Ø	13/16Ø	9/418

If there are to be more file systems mounted than just the root ,/sys, and /usr, use mkfs(1) to create any new file system and put its mounting in the file /etc/rc (see init(8) and mount(1)). (You might look at /etc/rc anyway to see what has been provided for you.)

There are two considerations in deciding how to adjust the arrangement of things on your disks: the most important is making sure there is adequate space for what is required; secondarily, throughput should be maximized. Swap space is a critical parameter. The system as distributed has 8778 (hpunix), 8778 (hkunix), 8ØØØ (rpunix) , 3ØØØ (rlØlunix), 4ØØØ (rlØ2unix) blocks for swap space. This should be large enough so running out of swap space never occurs. You may want to change these if local wisdom indicates otherwise.

Many common system programs (C, the editor, the assembler etc.) create intermediate files in the /tmp directory, so the file system where this is stored also should be made large enough to accommodate most high-water marks. If you leave the root file system as distributed (except as discussed above) there should be no problem. All the programs that create files in /tmp take care to delete them, but most are not immune to events like being hung up upon, and can leave dregs. The directory should be examined every so often and the old files deleted.

Exhaustion of user-file space is certain to occur now and then; the only mechanisms for controlling this phenomenon are occasional use of du(1), df(1), quot(1), threatening messages of the day, and personal letters.

The efficiency with which UNIX is able to use the CPU is largely dictated by the configuration of disk controllers. For general time-sharing applications, the best strategy is to try to split user files, the root directory (including the /tmp directory) and the swap area among three controllers.

Once you have decided how to make best use of your hardware, the question is how to initialize it. If you have the equipment, the best way to move a file system is to dump it (dump(1)) to magtape, use mkfs(1) to create the new file system, and restore (restor(1)) the tape. If for some reason you don't want to use magtape, dump accepts an argument telling where to put the dump; you might use another disk. Sometimes a file system has to be increased in logical size without copying. The super-block of the device has a word giving the highest address which can be allocated. For relatively small increases, this word can be patched using the debugger (adb(1)) and the free list reconstructed using icheck(1). The size should not be increased very greatly by this technique, however, since although the allocatable space will increase the maximum number of files will not (that is, the i-list size can't be changed). Read and understand the description given in file system(5) before playing around in this way. You may want to see section rp(4) or hp(4) for some suggestions on how to lay out the information on RP disks. More detailed information about the disk partitions may be obtained from the `sizes' tables in the disk drivers (hp.c, hk.c, rp.c). Also see /usr/doc/hksizes and /usr/doc/hpsizes for the RK06/7 and RM02/3, RP04/5/6 disk layouts.

If you have to merge a file system into another, existing one, the best bet is to use tar(1). If you must shrink a file system, the best bet is to dump the original and restor it onto the new filesystem. However, this might not work if the i-list on the smaller filesystem is smaller than the maximum allocated inode on the larger. If this is the case, reconstruct the filesystem from scratch on another filesystem (perhaps using tar(1)) and then dump it. If you are playing with the root file system and only have one drive the procedure is more complicated. What you do is the following:

1. GET A SECOND PACK!!!!

2. Dump the current root filesystem (or the reconstructed one) using dump(1).

3. Bring the system down and mount the new pack.

4. Retrieve the WECo distribution tape and perform steps 1 through 5 at the beginning of this document, substituting the desired file system size instead of 9600 when asked for `file system size'.

5. Perform step 6 above up to the point where the `tape' question is asked. At this point mount the tape you made just a few minutes ago. Continue with step 6 above substituting a 0 (zero) for the 5.

New Users

Install new users by editing the password file /etc/passwd (passwd(5)). This procedure should be done once multi-user mode is entered (see init(8)). You'll have to make a current directory for each new user and change its owner to the newly installed name. Login as each user to make sure the password file is correctly edited. For example:

```
ed /etc/passwd
$a
joe::10:1::/usr/joe:
.
w
q
mkdir /usr/joe
chown joe /usr/joe
login joe
ls -la
login root
```

This will make a new login entry for joe, who should be encouraged to use passwd(1) to give himself a password. His default current directory is /usr/joe which has been created. The delivered password file has the user bin in it to be used as a prototype.

Multiple Users

If UNIX is to support simultaneous access from more than just the console terminal, the file /etc/ttys (ttys(5)) has to be edited. To add a new terminal be sure the device is configured and the special file exists, then set the first character of the appropriate line of /etc/ttys to 1 (or add a new line). If the new terminal is to be a "local" terminal, i.e., not a dialup line, set the first character to 2, this will enable the terminal to operate without having carriar asserted. Note that init.c will have to be recompiled if there are to be more than 100 terminals. Also note that if the special file is inaccessible when init tries to create a process for it, the system will thrash trying and retrying to open it.

File System Health

Periodically (say every day or so) and always after a crash, you should check all the file systems for consistency (fsck (1m)). You should create the file "/etc/checklist" containing the names of the file systems to be checked, see fsck (1m) for more detail. It is quite important to execute sync (1m) before rebooting or taking the machine down. This is done automatically every 30 seconds by the update program (8) when a multiple-user system is running, but you should do it anyway to make sure.

Dumping of the file system should be done regularly, since once the system is going it is very easy to become complacent. Complete and incremental dumps are easily done with dump(1). Dumping of files by name is best done by tar(1) but the number of files is somewhat limited. Finally if there are enough drives entire disks can be copied using cp(1), or preferably with dd(1) using the raw special files and an appropriate block size.

Converting Sixth Edition Filesystems

The best way to convert file systems from 6th edition (V6) to 7th edition (V7) format is to use tar(1). However, a special version of tar must be prepared to run on V6. The following steps will do this:

1. change directories to /usr/src/cmd/tar

2. At the shell prompt respond

```
make v6tar or make v6tar40
```

   This will leave an executable binary named `v6tar' or `v6tar40'.

3. Mount a scratch tape.

4. Use tp(1) to put `v6tar' on the scratch tape.

5. Bring down V7 and bring up V6.

6. Use tp (on V6) to read in `v6tar'. Put it in /bin or /usr/bin (or perhaps some other preferred location).

7. Use v6tar to make tapes of all that you wish to convert. You may want to read the manual section on tar(1) to see whether you want to use blocking or not. Try to avoid using full pathnames when making the tapes. This will simplify moving the hierarchy to some other place on V7 if desired. For example

```
chdir /usr/ken
v6tar c .
```

   is preferable to

```
v6tar c /usr/ken
```

8. After all of the desired tapes are made, bring down V6 and reboot V7. Use tar(1) to read in the tapes just made.

Odds and Ends

The programs dump, quot, ncheck, and df (source in /usr/source/cmd) should be changed to reflect your default mounted file system devices. Print the first few lines of these programs and the changes will be obvious. Tar should be changed to reflect your desired default tape drive.

Good Luck

Charles B. Haley
Dennis M. Ritchie
Fred Canter

Regenerating System Software

Charles B. Haley
Dennis. M. Ritchie

Bell Laboratories
Murray Hill, New Jersey 07974

This document has been updated to include the modifications to UNIX* version seven by the Unix Systems Engineering group at DIGITAL.

Unix/v7m Release 2.1

Fred Canter

Digital Equipment Corporation
Continential Boulevard
Merrimack, New Hampshire 03054

*UNIX is a Trademark of Bell Laboratories.

September 23, 1981

Introduction

This document discusses how to assemble or compile various parts of the unix system software. This may be necessary because a command or library is accidentally deleted or otherwise destroyed; also, it may be desirable to install a modified version of some command or library routine. A few commands depend to some degree on the current configuration of the system; thus in any new system modifications to some commands are advisable. Most of the likely modifications relate to the standard disk devices contained in the system. For example, the df(1) (`disk free') command has built into it the names of the standardly present disk storage drives (e.g. `/dev/rf0', `/dev/rp0'). Df(1) takes an argument to indicate which disk to examine, but it is convenient if its default argument is adjusted to reflect the ordinarily present devices. The companion document `Setting up UNIX' discusses which commands are likely to require changes. Several of the command sources `include' the file <sys/param.h>, it may be necessary to recompile some or all of these commands if the system tunable parameters are changed, see System Tuning below.

Where Commands and Subroutines Live

The source files for commands and subroutines reside in several subdirectories of the directory /usr/src. These subdirectories, and a general description of their contents, are

cmd	Source files for commands.
libc/stdio	Source files making up the `standard i/o package'.
libc/sys	Source files for the C system call interfaces.
libc/gen	Source files for most of the remaining routines described in section 3 of the manual.
libc/crt	Source files making up the C runtime support package, as in call save-return and long arithmetic.
libc/csu	Source for the C startup routines.
games	Source for (some of) the games. No great care has been taken to try to make it obvious how to compile these; treat it as a game.
libF77	Source for the Fortran 77 runtime library, exclusive of IO.
libI77	Source for the Fortran 77 IO runtime routines.
libdbm	Source for the `data-base manager' package dbm (3).
libfpsim	Source for the floating-point simulator routine.
libm	Source for the mathematical library.
libplot	Source for plotting routines.

Commands

The regeneration of most commands is straightforward. The `cmd' directory will contain either a source file for the command or a subdirectory containing the set of files that make up the command. If it is a single file the command

```
cd /usr/src/cmd
cmake cmd_name
```

suffices. (Cmd_name is the name of the command you are playing with.) The result of the cmake command will be an executable version. If you type

```
cmake -cp cmd_name
```

the result will be copied to /bin (or perhaps /etc or other places if appropriate).

The cmake command has been modified to make the alternate version of certian commands required for operation of unix version seven on the PDP11/23, PDP11/24, PDP11/34, PDP11/40, and PDP11/60 processors. These commands are compiled by

```
cmake cmd_name40
```

and are named as follows:

```
dcheck40
dump40
dumpdir40
icheck40
ncheck40
restor40
```

Prior to making a command the correct version of the parameter file must be copied to /usr/include/sys/param.h, use /sys/h/param_ov.h for non separate I and D space CPUs and /sys/h/param_id.h for separate I and D space CPUs. In order to simplify this process a `makefile' has been provided, which automatically selects the correct parameter file, recompiles all necessary commands, and installs them in /bin or /etc as cmd_name40 or cmd_name70. For non separate I and D space CPUs use:

```
cd /usr/src/cmd
make cmd40
```

For separate I and D space CPUs use:

```
cd /usr/src/cmd
make cmd70
```

The command `make all' can be used to make both cmd40 and cmd70. The `makefile' in /bin can then be used to copy desired version of each command to its normal name, i.e., dump40 or dump70 to dump, etc.

If the source files are in a subdirectory there will be a `makefile' (see make(1)) to control the regeneration. After changing to the proper directory (cd(1)) you type one of the following:

make all — The program is compiled and loaded; the executable is left in the current directory.

make cp — The program is compiled and loaded, and the executable is installed. Everything is cleaned up afterwards; for example .o files are deleted.

make cmp — The program is compiled and loaded, and the executable is compared against the one in /bin.

Some of the makefiles have other options. Print (cat(1)) the ones you are interested in to find out.

The makefile for the tar command has been updated to make tar40, which is requried for non separate I & D space processors.

There are now six versions of `adb', in order to deal with the various types of unix kernels and the absence of floating point hardware. Refer to the `README' file in `/usr/src/cmd/adb' for information on how to select the appropriate version of adb and how to generate it.

The Assembler

The assembler consists of two executable files: /bin/as and /lib/as2. The first is the Ø-th pass: it reads the source program, converts it to an intermediate form in a temporary file `/tmp/atmØ?', and estimates the final locations of symbols. It also makes two or three other temporary files which contain the ordinary symbol table, a table of temporary symbols (like 1:) and possibly an overflow intermediate file. The program /lib/as2 acts as an ordinary multiple pass assembler with input taken from the files produced by /bin/as.

The source files for /bin/as are named `/usr/src/cmd/as/as1?.s' (there are 9 of them); /lib/as2 is produced from the source files `/usr/src/cmd/as/as2?.s'; they likewise are 9 in number. Considerable care should be exercised in replacing either component of the assembler. Remember that if the assembler is lost, the only recourse is to replace it from some backup storage; a broken assembler cannot assemble itself.

There is now a second assembler `ovas', which is used by the overlay C compiler to generate the overlay text unix kernel. The `makefile' in `/usr/src/cmd/as' has been modified to make the overlay assembler. The file `/usr/src/cmd/as/README' contains more information on the overlay assembler.

The C Compiler

There is now an overlay C compiler, used to generate the overlay text unix kernel. The overlay C compiler and its make procedure have been integrated with the standard C compiler. The file `/usr/src/cmd/c/README' further explains the overlay C compiler and its generation.

The C compiler consists of seven routines: `/bin/cc', which calls the phases of the compiler proper, the compiler control line expander `/lib/cpp', the assembler (`as'), and the loader (`ld'). The phases of the C compiler are `/lib/cØ' or `/lib/ovcØ', which is the first phase of the compiler; `/lib/c1', which is the second phase of the compiler; and `/lib/c2', which is the optional third phase optimizer. The loss of the C compiler is as serious as that of the assembler.

The source for /bin/cc resides in `/usr/src/cmd/cc.c'. Its loss alone (or that of c2) is not fatal. If needed, prog.c can be compiled by

```
/lib/cpp prog.c >tempØ
/lib/cØ tempØ temp1 temp2
/lib/c1 temp1 temp2 temp3
as - temp3
ld -n /lib/crtØ.o a.out -lc
```

The source for the compiler proper is in the directory /usr/src/cmd/c. The first phase (/lib/cØ) or (/lib/ovcØ) is generated from the files cØØ.c, ..., cØ5.c, which must be compiled by the C compiler. There is also cØ.h, a header file included by the C programs of the first phase. To make a new /lib/cØ and /lib/ovcØ use

```
make -f mfnov cØ
make -f mfov ovcØ
```

Before installing the new cØs, it is prudent to save the old ones someplace.

The second phase of C (/lib/c1) is generated from the source files c1Ø.c, ..., c13.c, the include-file c1.h, and a set of object-code tables combined into table.o. To generate a new second phase use

```
make -f mfov c1
```

It is likewise prudent to save c1 before installing a new version. In fact in general it is wise to save the object files for the C compiler so that if disaster strikes C can be reconstituted without a working version of the compiler.

In a similar manner, the third phase of the C compiler (/lib/c2) is made up from the files c2Ø.c and c21.c together with c2.h, and is compiled by the command:

```
make -f mfov c2
```

Its loss is not critical since it is completely optional.

The set of tables mentioned above is generated from the file table.s. This `.s' file is not in fact assembler source; it must be converted by use of the cvopt program, whose source and object are located in the C directory. Normally this is taken care of by make(1). You might want to look at the makefile to see what it does.

UNIX

The source and object programs for UNIX are kept in the subdirectories of /sys. In the subdirectory h there are several files ending in `.h'; these are header files which are picked up (via `#include ...') as required by each system module. The file param.h contains the parameters for unix, there are two versions of this file, param_ov.h for the PDP11/23, PDP11/24, PDP11/34, PDP11/40, and PDP11/60 and param_id.h for the PDP11/44, PDP11/45, PDP11/55, and PDP11/70. The subdirectory dev consists mostly of the device drivers together with a few other things. The subdirectory sys is the rest of the system. The files LIB1_id in sys and LIB2_id in dev are archives (ar(1)) which contain the object versions of the routines in the directory, for the separate I & D space processors. The overlay text kernel object modules are in ovdev and the overlay system objects and a partial archive (LIB1_ov) are in ovsys.

Subdirectory conf contains the files which control device configuration of the system. L.s specifies the contents of the interrupt vectors; c.c contains the tables which relate device numbers to handler routines. A third file, mch_ov.s or mch_id.s , contains all the machine-language code in the system. A fourth file, mch0.s, is generated by mkconf(1) and contains flags indicating what sort of tape drive is available for taking crash dumps. It also specifies the device address of the tape controller used for crash dumps. The mch0.s file also contains a parameter for controlling the inclusion of floating point support in the machine language code.

The first step in the unix system generation process is to select the unix kernel that is most appropriate for your type of processor, there are three. The separate I & D space kernel `unix_id' is used with the PDP11/44, PDP11/45, PDP11/55, and PDP11/70 processors. The overlay text kernel `unix_ov' is used with the PDP11/23, PDP11/24, PDP11/34, PDP11/40, and PDP11/60 processors. The `unix_i' kernel is only used for the preconfigured unix systems needed to initially load unix onto the system disk from the distribution tape. After the type of kernel has been chosen, use the following procedure to make unix:

1. Examine the appropriate parameter file (param_ov.h or param_id.h), you will find several `#define ...' statements used to control the inclusion of various features in the kernel. The features are ACCT, FP, SEP_ID, DH, MX, UBUSMAP, PARITY, and LCKPHYS, their meanings are explained by the comments in the parameter file. Features are excluded by commenting out the define statements in the parameter file. The system tuning parameters, NBUF, NPROC and the like, are also in the parameter file. It is not advisable to modify these for the first system generation, the best thing to do is make unix for your configuration and test it, then experiment with the tuning parameters.

2. The device drivers contain statements defining the CSR address and the number of units to be supported. Check the drivers for the devices in your configuration to insure that these values are correct, edit the drivers if necessary.

3. You must insure that the `sys' and `dev' archives are up to date, the archives supplied with the system are current. If no changes to the drivers or the parameter file were made in steps 1 and 2, then no action is required. There are two methods of updating the archives (LIB1 and LIB2). The first is to recompile all the source files and recreate the archives as follows:

```
cd /sys/conf
make all??
```

where ?? is the CPU type, 23, 24, 34, 40, 44, 45, 55, 60, or 70. This would normally not be necessary unless the system tuning parameters in param.h are changed. The second method is to recompile only the source files that were chenged and rearchive them as follows:

```
cd /sys/conf
mksys_id file1 file2 ... file6
or
mkdev_id file1 file2 ... file6
```

for unix_id or

```
cd /sys/conf
mksys_ov file1 file2 ... file6
or
mkdev_ov file1 file2 ... file6
```

for unix_ov. As many as six source files may be recompiled at once, only the filename is typed, not the`.c'. These `mk' files automatically select the appropriate parameter file and copy it to param.h. For example if the hp and dz drivers were changed the commands would be:

```
cd /sys/conf
mkdev_id hp dz
```

those drivers will be recompiled and replaced in the LIB2_id archive. If the parameter file was changed in step 1, the comments in that file indicate which source files must be recompiled.

4. Prepare a configuration file, named `unixconf' or something like that, which describes your system configuration. Use mkconf(1m) and the many existing `conf' files in /sys/conf as a guide. If the overlay text kernel is to be used, the `conf' file must contain the `ov' declaration.

5. Run the mkconf program with the `conf' file as input:

```
mkconf <unixconf
```

   mkconf will print a list of the configured devices and their vectors.

6. Examine the core dump tape CSR address, in mchØ.s to verify that is matches your hardware. You may need to edit the low core vector file `l.s' to correct the device interrupt vectors, in any case examine `l.s' to insure that the vectors match your configuration. It is wise to print the configuration tables, in the file `c.c', and verify that the correct devices are entered in the bdevsw and cdevsw tables. You will need a copy of `c.c' later on anyway.

7. Use the `makefile', in /sys/conf to make unix as follows:

```
make unix??
```

   where ?? is the CPU type, 23, 24, 34, 4Ø, 44, 45, 55, 6Ø, 7Ø.

8. When the make is done, the new system is present in the current directory as `unix_ov' or `unix_id'. It should be tested before destroying the currently running `/unix', this is best done by doing something like

```
mv /unix /ounix
mv unix_ov /unix
```

   or

```
mv unix_id /unix
```

   You must be super-user to move unix to the root. If the new system doesn't work, you can still boot `ounix' and come up (see boot(8)). When you have satisfied yourself that the new system works, remove /ounix.

Installing new devices

Refer to mkconf(1m) and the `Unix/v7m Software Description' for information on what devices are supported by Unix/v7m. The information in this section is of general interest, however, the steps described below are only necessary if you need to add a new device that is not presently supported by mkconf(1m).

To install a new driver, compile it and put it into its library. The best way to put it into the library is to edit its name into the `mkdev' files in `/sys/conf' and the `mklib' files in /sys/dev, and then use `mkdev' to recompile and archive it. There is no LIB2 device driver library for the overlay kernel, `unix_ov'.

Next, the device's interrupt vector must be entered in l.s. This is probably already done by the routine mkconf(1), but if the device is esoteric or nonstandard you will have to massage l.s by hand. This involves placing a pointer to a callout routine and the device's priority level in the vector. Use some other device (like the console) as a guide. Notice that the entries in l.s must be in order as the assembler does not permit moving the location counter `.' backwards. The assembler also does not permit assignation of an absolute number to `.', which is the reason for the `. = ZERO+1ØØ' subterfuge. If a constant smaller than 16(1Ø) is added to the priority level, this number will be available as the first argument of the interrupt routine. This stratagem is used when several similar devices share the same interrupt routine (as in dlll's).

If you have to massage l.s, be sure to add the code to actually transfer to the interrupt routine. Again use the console as a guide. The apparent strangeness of this code is due to running the kernel in separate I&D space. The call routine saves registers as required and prepares a C-style call on the actual interrupt routine named after the `jmp' instruction. When the routine returns, call restores the registers and performs an rti instruction. As an aside, note that external names in C programs have an underscore (`_') prepended to them.

The second step which must be performed to add a device unknown to mkconf is to add it to the configuration table /sys/conf/c.c. This file contains two subtables, one for block-type devices, and one for character-type devices. Block devices include disks, DECtape, and magtape. All other devices are character devices. A line in each of these tables gives all the information the system needs to know about the device handler; the ordinal position of the line in the table implies its major device number, starting at 0.

There are four subentries per line in the block device table, which give its open routine, close routine, strategy routine, and device table. The open and close routines may be nonexistent, in which case the name `nulldev' is given; this routine merely returns. The strategy routine is called to do any I/O, and the device table contains status information for the device.

For character devices, each line in the table specifies a routine for open, close, read, and write, and one which sets and returns device-specific status (used, for example, for stty and gtty on typewriters). 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.

The final step which must be taken to install a device is to make a special file for it. This is done by mknod(1), to which you must specify the device class (block or character), major device number (relative line in the configuration table) and minor device number (which is made available to the driver at appropriate times).

The documents `Setting up Unix' and `The Unix IO system' may aid in comprehending these steps.

The Library libc.a

The library /lib/libc.a is where most of the subroutines described in sections 2 and 3 of the manual are kept. This library can be remade using the following commands:

```
cd /usr/src/libc
sh compall
sh mklib
mv libc.a /lib/libc.a
```

If single routines need to be recompiled and replaced, use

```
cc -c -O x.c
ar vr /lib/libc.a x.o
rm x.o
```

The above can also be used to put new items into the library. See ar(1), lorder(1), and tsort(1).

The routines in /usr/src/cmd/libc/csu (C start up) are not in libc.a. These are separately assembled and put into /lib. The commands to do this are

```
cd /usr/src/libc/csu
as - x.s
mv a.out /lib/x
```

where x is the routine you want.

Other Libraries

Likewise, the directories containing the source for the other libraries have files compall (that recompiles everything) and mklib (that recreates the library).

System Tuning

There are several tunable parameters in the system. These set the size of various tables and limits. They are found in the file /sys/h/param.h as manifests (`#define's), remember that there are two versions of this file, param_ov.h and param_id.h. Their values are rather generous in the system as distributed. Our typical maximum number of users is about 20, but there are many daemon processes. The values of the parameters in the param_ov.h file are set for about 10 users.

When any parameter is changed, it is prudent to recompile the entire system, as discussed above. A brief discussion of each follows:

NBUF — This sets the size of the disk buffer cache. Each buffer is 512 bytes. This number should be around 25 plus NMOUNT, or as big as can be if the above number of buffers cause the system to not fit in memory.

NFILE — This sets the maximum number of open files. An entry is made in this table every time a file is `opened' (see open(2), creat(2)). Processes share these table entries across forks (fork(2)). This number should be about the same size as NINODE below. (It can be a bit smaller.)

NMOUNT — This indicates the maximum number of mounted file systems. Make it big enough that you don't run out at inconvenient times.

MAXMEM — This sets an administrative limit on the amount of memory a process may have. It is set automatically if the amount of physical memory is small, and thus should not need to be changed.

MAXUPRC — This sets the maximum number of processes that any one user can be running at any one time. This should be set just large enough that people can get work done but not so large that a user can hog all the processes available (usually by accident!).

NPROC — This sets the maximum number of processes that can be active. It depends on the demand pattern of the typical user; we seem to need about 8 times the number of terminals.

NINODE This sets the size of the inode table. There is one entry in the inode table for every open device, current working directory, sticky text segment, open file, and mounted device. Note that if two users have a file open there is still only one entry in the inode table. A reasonable rule of thumb for the size of this table is

NPROC + NMOUNT + (number of terminals)

SSIZE The initial size of a process stack. This may be made bigger if commonly run processes have large data areas on the stack.

SINCR The size of the stack growth increment.

NOFILE This sets the maximum number of files that any one process can have open. 20 is plenty.

CANBSIZ This is the size of the typewriter canonicalization buffer. It is in this buffer that erase and kill processing is done. Thus this is the maximum size of an input typewriter line. 256 is usually plenty.

CMAPSIZ The number of fragments that memory can be broken into. This should be big enough that it never runs out. This parameter automatically grows as NPROC is increased.

SMAPSIZ Same as CMAPSIZ except for secondary (swap) memory.

NCALL This is the size of the callout table. Callouts are entered in this table when some sort of internal system timing must be done, as in carriage return delays for terminals. The number must be big enough to handle all such requests.

NTEXT The maximum number of simultaneously executing pure programs. This should be big enough so as to not run out of space under heavy load. A reasonable rule of thumb is about

(number of terminals) + (number of sticky programs)

NCLIST The number of clist segments. A clist segment is 6 characters. NCLIST should be big enough so that the list doesn't become exhausted when the machine is busy. The characters that have arrived from a terminal and are waiting to be given to a process live here. Thus enough space should be left so that every terminal can have

at least one average line pending (about 30 or 40 characters).

TIMEZONE — The number of minutes westward from Greenwich. See `Setting Up UNIX'.

DSTFLAG — See `Setting Up UNIX' section on time conversion.

MSGBUFS — The maximum number of characters of system error messages saved. This is used as a circular buffer.

NCARGS — The maximum number of characters in an exec(2) arglist. This number controls how many arguments can be passed into a process. 5120 is practically infinite.

HZ — Set to the frequency of the system clock (e.g., 50 for a 50 Hz. clock).

System tuning on non separate I & D space CPUs

The overlay text unix kernel is used for the non separate I & D space processors, PDP11/23, PDP11/24, PDP11/34, PDP11/40, and PDP11/60. The system tuning parameters are set for about 10 users, as follows:

```
NBUF    = 14
NINODE  = 100
NFILE   = 80
NPROC   = 70
NTEXT   = 25
NCLIST  = 125
```

The following table can be used as an aid when tuning unix version seven on the non separate I & D space CPUs. It lists the name of the paramter, the size increase in bytes of incrementing the parameter by one, and the source files which must be recompiled if the parameter is changed.

PARAMETER	SIZE	FILES
NBUF	542	c.c, bio.c , main.c
NINODE	74	c.c, alloc.c, iget.c, sys3.c
NFILE	8	c.c, mx2.c, fio.c, iget.c
NMOUNT	6	c.c, alloc.c, iget.c, nami.c, sys3.c (one system buffer per NMOUNT)
MAXUPRC	0	c.c, sysl.c
NOFILE	0	c.c, fio.c, slp.c, sysl.c, sys3.c
CMAPSIZ	4	c.c, (any file that `includes' map.h)
SMAPSIZ	4	c.c, (any file that `includes' map.h)
NCALL	6	c.c, clock.c
NPROC	28	c.c
NTEXT	12	c.c, text.c
NCLIST	8	c.c, prim.c (clists are larger on I & D space CPUs)

The size of the overlay text kernel can be reduced by deselecting unneeded features in the param_ov.h, rebuilding the sys and dev archives, and remaking unix, as described in the section on generating UNIX above.

NAME

adb - debugger

SYNOPSIS

adb [-w] [objfil [corfil]]

DESCRIPTION

Adb is a general purpose debugging program. It may be used to examine files and to provide a controlled environment for the execution of UNIX programs. There are now multiple versions of adb , in order to support the three types of UNIX systems currently available. The appropriate version of adb is selected at sysgen time, see "Setting up Unix" for more on this.

Objfil is normally an executable program file, preferably containing a symbol table; if not then the symbolic features of adb cannot be used although the file can still be examined. The default for objfil is a.out. Corfil is assumed to be a core image file produced after executing objfil ; the default for corfil is core.

Requests to adb are read from the standard input and responses are to the standard output. If the -w flag is present then both objfil and corfil are created if necessary and opened for reading and writing so that files can be modified using adb . Adb ignores QUIT; INTERRUPT causes return to the next adb command.

In general requests to adb are of the form

[address] [, count] [command] [;]

If address is present then dot is set to address . Initially dot is set to 0. For most commands count specifies how many times the command will be executed. The default count is 1. Address and count are expressions.

The interpretation of an address depends on the context it is used in. If a subprocess is being debugged then addresses are interpreted in the usual way in the address space of the subprocess. For further details of address mapping see ADDRESSES.

EXPRESSIONS

. The value of dot .

+ The value of dot incremented by the current increment.

^ The value of dot decremented by the current increment.

" The last address typed.

integer
An octal number if integer begins with a Ø; a hexadecimal number if preceded by #; otherwise a decimal number.

integer.fraction
A 32 bit floating point number.

'cccc' The ASCII value of up to 4 characters. \ may be used to escape a '.

< name The value of name , which is either a variable name or a register name. Adb maintains a number of variables (see VARIABLES) named by single letters or digits. If name is a register name then the value of the register is obtained from the system header in corfil . The register names are rØ ... r5 sp pc ps.

symbol A symbol is a sequence of upper or lower case letters, underscores or digits, not starting with a digit. The value of the symbol is taken from the symbol table in objfil . An initial _ or ~ will be prepended to symbol if needed.

_ symbol
In C, the `true name' of an external symbol begins with _. It may be necessary to utter this name to disinguish it from internal or hidden variables of a program.

routine.name
The address of the variable name in the specified C routine. Both routine and name are symbols . If name is omitted the value is the address of the most recently activated C stack frame corresponding to routine .

(exp) The value of the expression exp .

Monadic operators

*exp The contents of the location addressed by exp in corfil .

@exp The contents of the location addressed by exp in objfil .

-exp Integer negation.

~exp Bitwise complement.

Dyadic operators are left associative and are less binding than monadic operators.

e1 + e2
Integer addition.

e1 - e2
Integer subtraction.

e1 * e2
Integer multiplication.

e1 % e2
Integer division.

e1 & e2
Bitwise conjunction.

e1 | e2
Bitwise disjunction.

e1 # e2
E1 rounded up to the next multiple of e2 .

COMMANDS

Most commands consist of a verb followed by a modifier or list of modifiers. The following verbs are available. (The commands `?' and `/' may be followed by `*'; see ADDRESSES for further details.)

?f Locations starting at address in objfil are printed according to the format f .

/f Locations starting at address in corfil are printed according to the format f .

=f The value of address itself is printed in the styles indicated by the format f . (For i format `?' is printed for the parts of the instruction that reference subsequent words.)

A format consists of one or more characters that specify a style of printing. Each format character may be preceded by a decimal integer that is a repeat count for the format character. While stepping through a format dot is incremented temporarily by the amount given for each format letter. If no format is given then the last format is used. The format letters available are as follows.

o 2 Print 2 bytes in octal. All octal numbers output by adb are preceded by 0.
O 4 Print 4 bytes in octal.

q 2 Print in signed octal.
Q 4 Print long signed octal.
d 2 Print in decimal.
D 4 Print long decimal.
x 2 Print 2 bytes in hexadecimal.
X 4 Print 4 bytes in hexadecimal.
u 2 Print as an unsigned decimal number.
U 4 Print long unsigned decimal.
f 4 Print the 32 bit value as a floating point number.
F 8 Print double floating point.
b 1 Print the addressed byte in octal.
c 1 Print the addressed character.
C 1 Print the addressed character using the following escape convention. Character values 000 to 040 are printed as @ followed by the corresponding character in the range 0100 to 0140. The character @ is printed as @@.
s n Print the addressed characters until a zero character is reached.
S n Print a string using the @ escape convention. n is the length of the string including its zero terminator.
Y 4 Print 4 bytes in date format (see ctime (3)).
i n Print as PDP11 instructions. n is the number of bytes occupied by the instruction. This style of printing causes variables 1 and 2 to be set to the offset parts of the source and destination respectively.
a 0 Print the value of dot in symbolic form. Symbols are checked to ensure that they have an appropriate type as indicated below.

/ local or global data symbol
? local or global text symbol
= local or global absolute symbol

p 2 Print the addressed value in symbolic form using the same rules for symbol lookup as a.
t 0 When preceded by an integer tabs to the next appropriate tab stop. For example, 8t moves to the next 8-space tab stop.
r 0 Print a space.
n 0 Print a newline.
"..." 0
Print the enclosed string.
^ Dot is decremented by the current increment. Nothing is printed.
+ Dot is incremented by 1. Nothing is printed.
- Dot is decremented by 1. Nothing is printed.

newline
If the previous command temporarily incremented dot ,

make the increment permanent. Repeat the previous command with a count of 1.

[?/]l value mask
Words starting at dot are masked with mask and compared with value until a match is found. If L is used then the match is for 4 bytes at a time instead of 2. If no match is found then dot is unchanged; otherwise dot is set to the matched location. If mask is omitted then -1 is used.

[?/]w value ...
Write the 2-byte value into the addressed location. If the command is W, write 4 bytes. Odd addresses are not allowed when writing to the subprocess address space.

[?/]m b1 e1 f1[?/]
New values for (b1, e1, f1) are recorded. If less than three expressions are given then the remaining map parameters are left unchanged. If the `?' or `/' is followed by `*' then the second segment (b2,e2,f2) of the mapping is changed. If the list is terminated by `?' or `/' then the file (objfil or corfil respectively) is used for subsequent requests. (So that, for example, `/m?' will cause `/' to refer to objfil .)

>name
Dot is assigned to the variable or register named.

!
A shell is called to read the rest of the line following `!'.

$modifier
Miscellaneous commands. The available modifiers are:

- <f Read commands from the file f and return.
- >f Send output to the file f, which is created if it does not exist.
- r Print the general registers and the instruction addressed by pc. Dot is set to pc.
- f Print the floating registers in single or double length. If the floating point status of ps is set to double (0200 bit) then double length is used anyway.
- b Print all breakpoints and their associated counts and commands.
- a ALGOL 68 stack backtrace. If address is given then it is taken to be the address of the current frame (instead of r4). If count is given then only the first count frames are printed.
- c C stack backtrace. If address is given then it is taken as the address of the current frame (instead

of r5). If C is used then the names and (16 bit) values of all automatic and static variables are printed for each active function. If count is given then only the first count frames are printed.

e The names and values of external variables are printed.

w Set the page width for output to address (default 80).

s Set the limit for symbol matches to address (default 255).

o All integers input are regarded as octal.

d Reset integer input as described in EXPRESSIONS.

q Exit from adb .

v Print all non zero variables in octal.

m Print the address map.

:modifier

Manage a subprocess. Available modifiers are:

bc Set breakpoint at address . The breakpoint is executed count -1 times before causing a stop. Each time the breakpoint is encountered the command c is executed. If this command sets dot to zero then the breakpoint causes a stop.

d Delete breakpoint at address .

r Run objfil as a subprocess. If address is given explicitly then the program is entered at this point; otherwise the program is entered at its standard entry point. count specifies how many breakpoints are to be ignored before stopping. Arguments to the subprocess may be supplied on the same line as the command. An argument starting with < or > causes the standard input or output to be established for the command. All signals are turned on on entry to the subprocess.

cs The subprocess is continued with signal s c s, see signal (2). If address is given then the subprocess is continued at this address. If no signal is specified then the signal that caused the subprocess to stop is sent. Breakpoint skipping is the same as for r.

ss As for c except that the subprocess is single stepped count times. If there is no current subprocess then objfil is run as a subprocess as for r. In this case no signal can be sent; the remainder of the line is treated as arguments to the subprocess.

k The current subprocess, if any, is terminated.

VARIABLES

Adb provides a number of variables. Named variables are set initially by adb but are not used subsequently. Numbered variables are reserved for communication as follows.

Ø The last value printed.
1 The last offset part of an instruction source.
2 The previous value of variable 1.

On entry the following are set from the system header in the corfil . If corfil does not appear to be a core file then these values are set from objfil .

b The base address of the data segment.
d The data segment size.
e The entry point.
m The `magic' number (Ø4Ø5, Ø4Ø7, Ø41Ø or Ø411).
s The stack segment size.
t The text segment size.

ADDRESSES

The address in a file associated with a written address is determined by a mapping associated with that file. Each mapping is represented by two triples (b1, e1, f1) and (b2, e2, f2) and the file address corresponding to a written address is calculated as follows.

b1 ≤ address < e1 => file address = address + f1-b1, otherwise,

b2 ≤ address < e2 => file address = address + f2-b2,

otherwise, the requested address is not legal. In some cases (e.g. for programs with separated I and D space) the two segments for a file may overlap. If a ? or / is followed by an * then only the second triple is used.

The initial setting of both mappings is suitable for normal a.out and core files. If either file is not of the kind expected then, for that file, b1 is set to Ø, e1 is set to the maximum file size and f1 is set to Ø; in this way the whole file can be examined with no address translation.

So that adb may be used on large files all appropriate values are kept as signed 32 bit integers.

FILES

/dev/mem
/dev/swap
a.out

core

SEE ALSO

ptrace(2), a.out(5), core(5), Setting up Unix

DIAGNOSTICS

`Adb' when there is no current command or format. Comments about inaccessible files, syntax errors, abnormal termination of commands, etc. Exit status is 0, unless last command failed or returned nonzero status.

BUGS

A breakpoint set at the entry point is not effective on initial entry to the program.
When single stepping, system calls do not count as an executed instruction.
Local variables whose names are the same as an external variable may foul up the accessing of the external.
The debugger is not fully able to deal with the overlay text Unix kernel. Only symbols in the root text segment and in the data and bss segments can be accessed by adb.

NAME

as - assembler
ovas - overlay assembler

SYNOPSIS

as [-] [-o objfile] file ...
ovas [-] [-o objfile] file ...

DESCRIPTION

As assembles the concatenation of the named files. If the optional first argument - is used, all undefined symbols in the assembly are treated as global.

The output of the assembly is left on the file objfile; if that is omitted, a.out is used. It is executable if no errors occurred during the assembly, and if there were no unresolved external references.

The overlay assembler is only used for the generation of the overlay text unix kernel.

FILES

/lib/as2	pass 2 of the assembler
/lib/ovas2	pass 2 of the overlay assembler
/tmp/atm[1-3]?	temporary
a.out	object

SEE ALSO

ld(1), nm(1), adb(1), a.out(5)
UNIX Assembler Manual by D. M. Ritchie

DIAGNOSTICS

When an input file cannot be read, its name followed by a question mark is typed and assembly ceases. When syntactic or semantic errors occur, a single-character diagnostic is typed out together with the line number and the file name in which it occurred. Errors in pass 1 cause cancellation of pass 2. The possible errors are:

)	Parentheses error
]	Parentheses error
<	String not terminated properly
*	Indirection used illegally
.	Illegal assignment to `.'
a	Error in address
b	Branch instruction is odd or too remote
e	Error in expression
f	Error in local (`f' or `b') type symbol
g	Garbage (unknown) character
i	End of file inside an if
m	Multiply defined symbol as label
o	Word quantity assembled at odd address

p `.' different in pass 1 and 2
r Relocation error
u Undefined symbol
x Syntax error

BUGS

Syntax errors can cause incorrect line numbers in following diagnostics.

NAME

cc, pcc - C compiler

SYNOPSIS

cc [option] ... file ...

pcc [option] ... file ...

DESCRIPTION

Cc is the UNIX C compiler. It accepts several types of arguments:

Arguments whose names end with `.c' are taken to be C source programs; they are compiled, and each object program is left on the file whose name is that of the source with `.o' substituted for `.c'. The `.o' file is normally deleted, however, if a single C program is compiled and loaded all at one go.

In the same way, arguments whose names end with `.s' are taken to be assembly source programs and are assembled, producing a `.o' file.

The following options are interpreted by cc . See ld (1) for load-time options.

-c Suppress the loading phase of the compilation, and force an object file to be produced even if only one program is compiled.

-p Arrange for the compiler to produce code which counts the number of times each routine is called; also, if loading takes place, replace the standard startup routine by one which automatically calls monitor (3) at the start and arranges to write out a mon.out file at normal termination of execution of the object program. An execution profile can then be generated by use of prof (1).

-f In systems without hardware floating-point, use a version of the C compiler which handles floating-point constants and loads the object program with the floating-point interpreter. Do not use if the hardware is present.

-O Invoke an object-code optimizer.

-S Compile the named C programs, and leave the assembler-language output on corresponding files suffixed `.s'.

-P Run only the macro preprocessor and place the result

for each `.c' file in a corresponding `.i' file and has no `#' lines in it.

-E Run only the macro preprocessor and send the result to the standard output. The output is intended for compiler debugging; it is unacceptable as input to cc .

-V Invoke the overlay version of the C compiler. This option is only used for the generation of the overlay text unix kernel.

-o output
Name the final output file output . If this option is used the file `a.out' will be left undisturbed.

-Dname=def
-Dname Define the name to the preprocessor, as if by `#define'. If no definition is given, the name is defined as 1.

-Uname Remove any initial definition of name .

-Idir `#include' files whose names do not begin with `/' are always sought first in the directory of the file argument, then in directories named in -I options, then in directories on a standard list.

-Bstring
Find substitute compiler passes in the files named string with the suffixes cpp, cØ, c1 and c2. If string is empty, use a standard backup version.

-t[pØ12]
Find only the designated compiler passes in the files whose names are constructed by a -B option. In the absence of a -B option, the string is taken to be `/usr/c/'.

Other arguments are taken to be either loader option arguments, or C-compatible object programs, typically produced by an earlier cc run, or perhaps libraries of C-compatible routines. These programs, together with the results of any compilations specified, are loaded (in the order given) to produce an executable program with name a.out.

The major purpose of the `portable C compiler', pcc, is to serve as a model on which to base other compilers. Pcc does not support options -f, -E, -B, and -t. It provides, in addition to the language of cc, unsigned char type data and initialized bit fields.

FILES

file.c	input file
file.o	object file
a.out	loaded output
/tmp/ctm?	temporaries for cc
/lib/cpp	preprocessor
/lib/c[Ø1]	compiler for cc
/lib/ovcØ	overlay compiler for cc
/usr/c/oc[Ø12]	backup compiler for cc
/usr/c/ocpp	backup preprocessor
/lib/fc[Ø1]	floating-point compiler
/lib/c2	optional optimizer
/lib/crtØ.o	runtime startoff
/lib/mcrtØ.o	startoff for profiling
/lib/fcrtØ.o	startoff for floating-point interpretation
/lib/libc.a	standard library, see intro (3)
/usr/include	standard directory for `#include' files
/tmp/pc*	temporaries for pcc
/usr/lib/ccom	compiler for pcc

SEE ALSO

B. W. Kernighan and D. M. Ritchie, The C Programming Language, Prentice-Hall, 1978
D. M. Ritchie, C Reference Manual
monitor(3), prof(1), adb(1), ld(1)

DIAGNOSTICS

The diagnostics produced by C itself are intended to be self-explanatory. Occasional messages may be produced by the assembler or loader. Of these, the most mystifying are from the assembler, as (1), in particular `m', which means a multiply-defined external symbol (function or data).

BUGS

Pcc is little tried on the PDP11; specialized code generated for that machine has not been well shaken down. The -O optimizer was designed to work with cc ; its use with pcc is suspect.

NAME

fsck - file system consistency check and interactive repair

SYNOPSIS

/etc/fsck [option] ... [filesystem] ...] ...

DESCRIPTION

Fsck audits and interactively repairs inconsistent conditions for the named filesystems. 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 on PDP11/45 CPU
-s4 optimal for RP04, RP05, RP06 on PDP11/70 CPU
-sc:s space free blocks s blocks apart in cylinders of c blocks each.

Refer to `Setting up Unix', table 1 in `Disk Layout', for values of `c' and `s' to be used with other disk and CPU combinations. The value stated there as `m' should be used for `s' and the value for `n' as `c'.

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 files. 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 contains default list of file systems to check.

SEE ALSO

dcheck(1), icheck(1), filsys(5), crash(8), `Setting up Unix'

BUGS

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

NAME

iostat - report I/O statistics

SYNOPSIS

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

DESCRIPTION

Iostat delves into the system and reports certain statistics kept about input-output activity. Information is kept about up to six different disks (HP, HM, HK, ML, RP, RL) and about typewriters. For each disk drive, I/O 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 drive is examined and a tally is made if the disk drive is active. The processor state is also examined, this 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. The iostat reports are for all types of activity, seeks as well as data transfers, on all drives that have had any I/O activity since the system was booted, inactive and nonexistent drives are ignored.

The optional drive argument allows the reports to be limited to a specified subset of the available drives. Up to six drive names, of the form; hpØ, m11, r13, rp4, etc., may be specified. Reports will be generated for only those drives which exist and have been active.

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 controller was active (seeking or transferring, the percentage of time any diks drive was active, and the percentage of time spent in `IO wait:' idle, but

with a disk drive active.

-s Report the raw timing information for each active disk drive. The information consists of; the disk controller name and drive number, the disk's transfer rate (microseconds/word), the percentage of the total system time that the drive had I/O activity, the number of transfers on that drive, and the number of words transferred by the drive.

-b Report on the usage of I/O buffers. The report gives; the number of buffers in the pool, the number of buffered reads, number of read-ahead blocks, number of buffer cache hits, number of buffered writes, and the number of I/O operations on each buffer starting with the first one.

-d Print the date and time at the head of the report.

-a Print the total time in minutes at the end of the report.

FILES

/dev/mem, /unix

BUGS

The accuracy of the iostat reports is subject to the sixtieth of a second granularity of the system clock.

NAME

ld - loader
covld - overlay loader

SYNOPSIS

ld [option] file ...
covld [option] file ...

DESCRIPTION

Ld combines several object programs into one, resolves external references, and searches libraries. In the simplest case several object files are given, and ld combines them, producing an object module which can be either executed or become the input for a further l run. (In the latter case, the -r option must be given to preserve the relocation bits.) The output of ld is left on a.out. This file is made executable only if no errors occurred during the load.

The overlay loader is used only for the generation of the overlay text unix kernel.

The argument routines are concatenated in the order specified. The entry point of the output is the beginning of the first routine.

If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. If a routine from a library references another routine in the library, and the library has not been processed by ranlib (l), the referenced routine must appear after the referencing routine in the library. Thus the order of programs within libraries may be important. If the first member of a library is named `__.SYMDEF', then it is understood to be a dictionary for the library such as produced by ranlib ; the dictionary is searched iteratively to satisfy as many references as possible.

The symbols `_etext', `_edata' and `_end' (`etext', `edata' and `end' in C) are reserved, and if referred to, are set to the first location above the program, the first location above initialized data, and the first location above all data respectively. It is erroneous to define these symbols.

Ld understands several options. Except for -l, they should appear before the file names.

-s `Strip' the output, that is, remove the symbol table and relocation bits to save space (but impair the usefulness of the debugger). This information can also be removed by strip (1).

-u Take the following argument as a symbol and enter it as undefined in the symbol table. This is useful for loading wholly from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine.

-lx This option is an abbreviation for the library name `/lib/libx.a', where x is a string. If that does not exist, ld tries `/usr/lib/libx.a'. A library is searched when its name is encountered, so the placement of a -l is significant.

-x Do not preserve local (non-.globl) symbols in the output symbol table; only enter external symbols. This option saves some space in the output file.

-X Save local symbols except for those whose names begin with `L'. This option is used by cc (1) to discard internally generated labels while retaining symbols local to routines.

-r Generate relocation bits in the output file so that it can be the subject of another ld run. This flag also prevents final definitions from being given to common symbols, and suppresses the `undefined symbol' diagnostics.

-d Force definition of common storage even if the -r flag is present.

-n Arrange that when the output file is executed, the text portion will be read-only and shared among all users executing the file. This involves moving the data areas up to the first possible 4K word boundary following the end of the text.

-i When the output file is executed, the program text and data areas will live in separate address spaces. The only difference between this option and -n is that here the data starts at location Ø.

-o The name argument after -o is used as the name of the ld output file, instead of a.out.

-e The following argument is taken to be the name of the entry point of the loaded program; location Ø is the default.

-O This is an overlay file, only the text segment will be replaced by exec (2). Shared data must have the same layout as in the program overlaid.

-D The next argument is a decimal number that sets the size of the data segment.

FILES

/lib/lib*.a	libraries
/usr/lib/lib*.a	more libraries
a.out	output file

SEE ALSO

as(1), ar(1), cc(1), ranlib(1)

BUGS

NAME

logins - enable user logins
nologins - disable user logins

SYNOPSIS

logins
nologins

DESCRIPTION

The logins and nologins commands allow the super-user to selectively enable and disable user logins. The super-user may login as `root' on any terminal even when logins are disabled.

FILES

/etc/loglock disables user logins
/etc/sdloglock system shutdown in progress

SEE ALSO

login(1), shutdown(8)

DIAGNOSTICS

`No Logins', a user attempted to login while logins were disabled.

`LOGINS DISABLED', printed when the super-user logs in and user logins are disabled.

NAME

ls - list contents of directory

SYNOPSIS

ls [-ltasdrucifgp] name ...

DESCRIPTION

For each directory argument, ls lists the contents of the directory; for each file argument, ls repeats its name and any other information requested. The output is sorted alphabetically by default. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments appear before directories and their contents. There are several options:

-l List in long format, giving mode, number of links, owner, size in bytes, and time of last modification for each file. (See below.) If the file is a special file the size field will instead contain the major and minor device numbers.

-t Sort by time modified (latest first) instead of by name, as is normal.

-a List all entries; usually `.' and `..' are suppressed.

-s Give size in blocks, including indirect blocks, for each entry.

-d If argument is a directory, list only its name, not its contents (mostly used with -l to get status on directory).

-r Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.

-u Use time of last access instead of last modification for sorting (-t) or printing (-l).

-c Use time of last modification to inode (mode, etc.) instead of last modification to file for sorting (-t) or printing (-l).

-i Print i-number in first column of the report for each file listed.

-f Force each argument to be interpreted as a directory and list the name found in each slot. This option turns off -l, -t, -s, and -r, and turns on -a; the order is the order in which entries appear in the directory.

-g Give group ID instead of owner ID in long listing.

-p Print pathnames instead of just file names.

The mode printed under the -l option contains 11 characters which are interpreted as follows: the first character is

d if the entry is a directory;
b if the entry is a block-type special file;
c if the entry is a character-type special file;
- if the entry is a plain file.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to owner permissions; the next to permissions to others in the same user-group; and the last to all others. Within each set the three characters indicate permission respectively to read, to write, or to execute the file as a program. For a directory, `execute' permission is interpreted to mean permission to search the directory for a specified file. The permissions are indicated as follows:

r if the file is readable;
w if the file is writable;
x if the file is executable;
- if the indicated permission is not granted.

The group-execute permission character is given as s if the file has set-group-ID mode; likewise the user-execute permission character is given as s if the file has set-user-ID mode.

The last character of the mode (normally `x' or `-') is t if the 1000 bit of the mode is on. See chmod (1) for the meaning of this mode.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks is printed.

FILES

/etc/passwd to get user ID's for `ls -l'.
/etc/group to get group ID's for `ls -g'.

NAME

mkconf - generate configuration tables

SYNOPSIS

mkconf

DESCRIPTION

Mkconf examines a machine configuration table on its standard input. Its output is three files l.s , c.c and mchØ.s. The first is an assembler program that represents the interrupt vectors located in low memory addresses; the second contains initialized block and character device switch tables; the third is a header file for the machine language assist file (mch.s), which selects the tape to be used for core dumps and specifies whether or not floating point support is included in mch.s.

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

pc	(PC11)
lp	(LP11)
rf	(RS11)
hs	(RSØ3/RSØ4)
ml	(ML11)
tc	(TU56)
rk	(RKØ3/RKØ5)
rl	(RLØ1/2)
tm	(TU1Ø)
rp	(RPØ3)
hp	(RMØ2/3, RPØ4/5/6 on first RH)
hm	(RMØ2/3, RPØ4/5/6 on second RH)
hk	(RKØ6/7)
ht	(TU16)
ts	(TS11)
rx2	(RXØ2)
dc*	(DC11)
kl*	(KL11/DL11-ABC)
dl*	(DL11-E)
dp*	(DP11)
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 typewrite 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 hs and ml devices are mutually exclusive.

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.

dump dev addr
Selects the tape to be used as the core dump device. The dev specification is one of the three tapes ht, tm, or ts, and addr is an optional tape CSR address specification. If dump is ommitted, mkconf will make the core dump tape selection.

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

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

nfp Do not include floating point support in the machine language assist file, the default is to include floating point.

ov Causes mkconf to create the file "covld" which is used by the makefile in /sys/conf to link the overlay unix kernel. Must be specified if the overlay kernel is to be generated.

nsid Must be specified if a non-spearate I & D space unix system is to be generated. By default, a separate I & D space unix system is generated. If "ov" is specified, "nsid" is ignored by mkconf.

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

SEE ALSO

`Setting up Unix', in Volume 2.
`Regenerating System Software', in Volume 2.

BUGS

The set of devices it knows about, the set of drivers included, and the set of devices on the machine are mutually incomparable. Some handwork is certain to be necessary. 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/size [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, hharacter 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. The optional arguments m and n are the file system interleave factors , used for optimal free list spacing. If m and n are not specified, the values m = 3 and n = 500 are used. Refer to the "Setting up Unix" document , table 1 in `Disk Layout', for a list of the optimum values of m and n .

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), `Setting up Unix'

BUGS

There should be some way to specify links.

NAME

nm - print name list

SYNOPSIS

nm [-gnopru] [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. If the `a.out' file is an overlay text unix kernel, all test segment symbols are followed by their overlay text segment number. 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.

SEE ALSO

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

NAME

ps - process status

SYNOPSIS

ps [aklxvt#] [namelist]

DESCRIPTION

Ps prints certain indicia about active processes. The a option asks for information about all processes with terminals (ordinarily only one's own processes are displayed); x asks even about processes with no terminal; l asks for a long listing. The short listing contains the process ID, tty letter, the cumulative execution time of the process and an approximation to the command line. If v is given and l is not present, the sums of the child process's system and user times are printed following the cumulative execution time. The t option limits printouts to those processes associated with tty #. Specifing ? as the tty number to the t option will limit printouts to those processes not associated with a tty.

The long listing is columnar and contains

F Flags associated with the process. Ø1: in core; Ø2: system process; Ø4: locked in core (e.g. for physical I/O); 1Ø: being swapped; 2Ø: being traced by another process.

S The state of the process. Ø: nonexistent; S: sleeping; W: waiting; R: running; I: intermediate; Z: terminated; T: stopped.

UID The user ID of the process owner.

PID The process ID of the process; as in certain cults it is possible to kill a process if you know its true name.

PPID The process ID of the parent process.

CPU Processor utilization for scheduling.

PRI The priority of the process; high numbers mean low priority.

NICE Used in priority computation.

ADDR The core address of the process if resident, otherwise the disk address.

SZ The size in blocks of the core image of the process.

WCHAN
The event for which the process is waiting or sleeping; if blank, the process is running.

TTY The controlling tty for the process.

TIME The cumulative execution time for the process.

The command and its arguments.

A process that has exited and has a parent, but has not yet been waited for by the parent is marked <defunct>. Ps makes an educated guess as to the file name and arguments given when the process was created by examining core memory or the swap area. The method is inherently somewhat unreliable and in any event a process is entitled to destroy this information, so the names cannot be counted on too much.

If the k option is specified, the file /usr/sys/core is used in place of /dev/mem . This is used for postmortem system debugging. If a second argument is given, it is taken to be the file containing the system's namelist.

FILES

/unix	system namelist
/dev/mem	core memory
/usr/sys/core	alternate core file
/dev	searched to find swap device and tty names

SEE ALSO
kill(1)

BUGS
Things can change while ps is running; the picture it gives is only a close approximation to reality.
Some data printed for defunct processes is irrelevant

NAME

pstat - print system facts

SYNOPSIS

pstat [-aixptuf] [suboptions] [corefile] [namelist]

DESCRIPTION

Pstat interprets the contents of certain system tables. If corefile is given, the tables are sought there, otherwise in /dev/mem. The required namelist is taken from /unix , unless the optional namelist argument is given. If the namelist is specified then the corefile must also be specified. 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.

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:
- Ø 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:
- Ø1 loaded
- Ø2 the scheduler process
- Ø4 locked
- Ø1Ø swapped out
- Ø2Ø traced
- Ø4Ø used in tracing
- Ø1ØØ locked in by lock (2).

PRI Scheduling priority, see nice (2).
SIGNAL
Signals received (signals 1-16 coded in bits Ø-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 Ø.

-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
/unix namelist
/dev/mem default source of tables

SEE ALSO
ps(1), stat(2), filsys(5)
K. Thompson, UNIX Implementation

NAME

size - size of an object file

SYNOPSIS

size [object ...]

DESCRIPTION

Size prints the (decimal) number of bytes required by the text, data, and bss portions, and their sum in octal and decimal, of each object-file argument. For overlay text unix kernel files size prints the (decimal) number of bytes contained in the root text segment, each overlay text segment, the data and bss segments, the sum of the root, data, and bss segments, and the total text size. If no file is specified, a.out is used.

SEE ALSO

a.out(5)

NAME

strip - remove symbols and relocation bits

SYNOPSIS

strip name ...

DESCRIPTION

Strip removes the symbol table and relocation bits ordinarily attached to the output of the assembler and loader. This is useful to save space after a program has been debugged.

The effect of strip is the same as use of the -s option of ld .

Strip automatically accommodates overlay text unix kernel files.

FILES

/tmp/stm? temporary file

SEE ALSO

ld(1)

NAME

hk - RK611/RKØ6, RKØ7 moving head disk

DESCRIPTION

The octal representation of the minor device number is encoded idp , where i is an interleave flag, d is a physical , drive number, and p is a pseudodrive (subsection) within a physical unit. If i is Ø, the origins and sizes of the pseudodisks on each drive, counted in cylinders of 66 512-byte blocks, are:

disk	start	length
Ø	Ø	146
1	146	135
2	281	129
3	411	4Ø3
4	Ø	Ø
5	Ø	Ø
6	Ø	41Ø
7	Ø	814

If i is 1, the minor device consists of the specified pseudodisk on drives numbered Ø through the designated drive number. Successively numbered blocks are distributed across the drives in rotation.

Systems distributed for these devices use disk Ø for the root, disk 1 for swapping, disk 2 for the system sources, and disk 3 (rkØ7) or disk 6 (rkØ6 drive 1) for a mounted user file system. Disk 6 (RKØ6) or disk 7 (RKØ7) may be used to create a mounted user file system, which consists of an entire disk pack. Pseudodisks 6 and 7 should not be used on the system disk pack, because they cover the entire pack.

The block files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records.

A `raw' interface provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files conventionally begin with an extra `r.' In raw I/O the buffer must begin on a word boundary.

FILES

/dev/rp?, /dev/rrp?
/dev/hk?, /dev/rhk?

SEE ALSO

/usr/doc/hksizes

BUGS

In raw I/O read and write (2) truncate file offsets to 512-byte block boundaries, and write scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, read, write and lseek (2) should always deal in 512-byte multiples.

NAME

hp - RH-11/RMØ2, RPØ4, RPØ5, RPØ6 moving-head disk
- RH-7Ø/RMØ3, RPØ4, RPØ5, RPØ6 moving-head disk
- RH-11/ML11 solid state disk
- RH-7Ø/ML11 solid state disk

DESCRIPTION

The octal representation of the minor device number is encoded idp , where i is an interleave flag, d is a physical drive number, and p is a pseudodrive (subsection) within a physical unit. If i is Ø, the origins and sizes of the pseudodisks on each drive, counted in cylinders of 418 512-byte blocks, for the RPØ4/5/6 are:

disk	start	length
Ø	Ø	23
1	23	21
2	44	21
3	65	345
4	65	749
5	411	4Ø3
6	Ø	41Ø
7	Ø	814

The pseudodisks for the RMØ2/3 are:

disk	start	length
Ø	Ø	6Ø
1	6Ø	55
2	115	5Ø
3	165	657
4	Ø	Ø
5	Ø	Ø
6	Ø	Ø
7	Ø	822

If i is 1, the minor device consists of the specified pseudodisk on drives numbered Ø through the designated drive number. Successively numbered blocks are distributed across the drives in rotation.

Systems distributed for these devices use disk Ø for the root, disk 1 for swapping, disk 2 for the system souces, and disk 3 (rpØ4/5 & rmØ2/3), or disk 4 (rpØ6) for a mounted user file system.

The ML11 solid state disk may be connected to the RH11 or RH7Ø massbus disk controller in conjunction with RM and RP disks. The ML11 may be used for the swap device or perhaps mounted on /tmp. The ML11 has switch-selectable transfer rates of Ø.25 mb, Ø.5 mb, 1.Ø mb, and 2.Ø mb per second. The following transfer rate restrictions apply:

0.25 mb	all CPU's
0.5 mb	all CPU's
1.0 mb	PDP 11/70 with RH70 only
2.0 mb	NO PDP11 CPU's

The RM02/3, RP04/5/6, and ML11 disks may be attached to a second RH11 or RH70 massbus disk controller at the alternate address and vector. In this case these disks are referenced as hm? .

The block files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records.

A `raw' interface provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files conventionally begin with an extra `r.' In raw I/O the buffer must begin on a word boundary, and raw I/O to an interleaved device is likely to have disappointing results.

FILES

/dev/rp?, /dev/rrp?
/dev/hp?, /dev/rhp?
/dev/hm?, /dev/rhm?

SEE ALSO

/usr/doc/hpsizes

BUGS

In raw I/O read and write (2) truncate file offsets to 512-byte block boundaries, and write scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, read, write and lseek (2) should always deal in 512-byte multiples.

Raw device drivers don't work on interleaved devices.

NAME

hs - RH-11/RSØ3, RSØ4 fixed-head disk
- RH-7Ø/RSØ3, RSØ4 fixed-head disk

DESCRIPTION

The files hsØ ... hs7 refer to RSØ3 disk drives Ø through 7. The files hs8 ... hs15 refer to RSØ4 disk drives Ø through 7. The RSØ3 drives are each 1Ø24 blocks long and the RSØ4 drives are 2Ø48 blocks long.

The hs files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a `raw' interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw HS files begin with rhs. The same minor device considerations hold for the raw interface as for the normal interface. In raw I/O the buffer must begin on a word boundary.

FILES

/dev/hs?, /dev/rhs?

BUGS

In raw I/O read and write (2) truncate file offsets to 512-byte block boundaries, and write scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, read, write and lseek (2) should always deal in 512-byte multiples.

NAME

ml - RH-11/ML11 solid state disk
- RH-70/ML11 solid state disk

DESCRIPTION

ml? refers to an entire ML11 unit as a single sequentially addressed file. The size of each ML11 unit depends on the number of array modules installed. There are 512 512-byte blocks per array module, for a maximum size of 8192 blocks per unit. The ml disk driver requires that the ML11 be installed on a separate RH11 or RH70 massbus disk controller.

The ML11 may be used for the swap device or perhaps mounted on /tmp. The ML11 has switch-selectable transfer rates of 0.25 mb, 0.5 mb, 1.0 mb, and 2.0 mb per second. The following transfer rate restrictions apply:

0.25 mb	all CPU's
0.5 mb	all CPU's
1.0 mb	PDP 11/70 with RH70 only
2.0 mb	NO PDP11 CPU's

The block files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records.

A `raw' interface provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files conventionally begin with an extra `r.' In raw I/O the buffer must begin on a word boundary, and raw I/O to an interleaved device is likely to have disappointing results.

FILES

/dev/ml?, /dev/rml?

BUGS

In raw I/O read and write (2) truncate file offsets to 512-byte block boundaries, and write scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, read, write and lseek (2) should always deal in 512-byte multiples.

NAME

rl - RL11/RLØ1 or RLØ2 disk

DESCRIPTION

Rl? refers to an entire disk as a single sequentially-addressed file. Its 256-word blocks are numbered Ø to 1Ø239 (RLØ1) or Ø to 2Ø479 (RLØ2). The physical disk sector size is 128 words, however the logical block size is 256 words. Minor device numbers are drive numbers on one controller.

The rl files discussed above access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a `raw' interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw RL files begin with rrl and end with a number which selects the same disk as the corresponding rl file.

In raw I/O the buffer must begin on a word boundary, and counts should be a multiple of 512 bytes (a disk block). Likewise seek calls should specify a multiple of 512 bytes.

FILES

/dev/rl?, /dev/rrl?
/dev/rp?, /dev/rrp?

BUGS

In raw I/O read and write (2) truncate file offsets to 512-byte block boundaries, and write scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, read, write and lseek (2) should always deal in 512-byte multiples.

NAME

rp - RP-11/RP03 moving-head disk

DESCRIPTION

The files rp0 ... rp7 refer to sections of RP disk drive 0. The files rp8 ... rp15 refer to drive 1 etc. This allows a large disk to be broken up into more manageable pieces.

The origin and size of the pseudo-disks on each drive are as follows:

disk	start	length
0	0	9600
1	9600	8000
2	17600	7400
3	25000	55000
4	0	80000
5-7	unassigned	

Thus rp4 covers the whole drive, while rp0, rp1, rp3 can serve usefully as a root, swap, and mounted user file system respectively. The disk rp2 covers a mounted file system containing the system sources.

The rp files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a `raw' interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw RP files begin with rrp and end with a number which selects the same disk section as the corresponding rp file.

In raw I/O the buffer must begin on a word boundary.

FILES

/dev/rp?, /dev/rrp?

SEE ALSO

hp(4)

BUGS

In raw I/O read and write (2) truncate file offsets to 512-byte block boundaries, and write scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, read, write and lseek (2) should always deal in 512-byte multiples.

NAME

rx - RX02 floppy disk

DESCRIPTION

RX? refers to an entire disk as a single sequentially-addressed file. The physical disk sector size is 128 bytes for single density and 256 bytes for double density, the logical block size is 512 bytes. Each diskette has 500 logical blocks, single density and 1001 logical blocks, double density. The minor device numbers have the following significance:

name	minor device	unit	density
rx0	0	0	single
rx1	1	1	single
rx2	2	0	double
rx3	3	1	double

The rx files discussed above access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a `raw' interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw RX files begin with rrx and end with a number which selects the same disk as the corresponding rx file.

In raw I/O the buffer must begin on a word boundary, and counts should be a multiple of 512 bytes (a disk block). Likewise seek calls should specify a multiple of 512 bytes.

FILES

/dev/rx?, /dev/rrx?

BUGS

In raw I/O read and write (2) truncate file offsets to 512-byte block boundaries, and write scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, read, write and lseek (2) should always deal in 512-byte multiples.

NAME

ts - TS11 magtape interface

DESCRIPTION

The file mt1 refers to the DEC TS11 magtape. When opened for reading or writing, the tape is not rewound. When closed, it is rewound (unless the Ø2ØØ bit is on, see below). If the tape was open for writing, a double end-of-file is written. If the tape is not to be rewound the tape is backspaced to just between the two tapemarks.

A standard tape consists of a series of 512 byte records terminated by a double end-of-file. To the extent possible, the system makes it possible, if inefficient, to treat the tape like any other file. Seeks have their usual meaning and it is possible to read or write a byte at a time. Writing in very small units is inadvisable, however, because it tends to create monstrous record gaps.

The TS11 operates at 16ØØ BPI only and the driver does not support multiple units. If the Ø2ØØ bit is on (initial digit 2 or 3), the tape is not rewound on close. Note that the minor device number has no necessary connection with the file name, and in fact tp (1) turns the short name x into `/dev/mtx '.

The mt files discussed above are useful when it is desired to access the tape in a way compatible with ordinary files. When foreign tapes are to be dealt with, and especially when long records are to be read or written, the `raw' interface is appropriate. The associated file may be named rmt1 but the same minor-device considerations as for the regular files still apply.

Each read or write call reads or writes the next record on the tape. In the write case the record has the same length as the buffer given. During a read, the record size is passed back as the number of bytes read, provided it is no greater than the buffer size; if the record is long, an error is indicated. In raw tape I/O, the buffer must begin on a word boundary and the count must be even. Seeks are ignored. A zero count is returned when a tape mark is read; another read will fetch the first record of the next tape file.

FILES

/dev/mt1, /dev/rmt1, /dev/nrmt1

SEE ALSO

tp(1)

BUGS

In raw I/O, there should be a way to perform forward and backward record and file spacing and to write an EOF mark explicitly.

NAME

ttys - terminal initialization data

DESCRIPTION

The ttys file is read by the init program and specifies which terminal special files are to have a process created for them which will allow people to log in. It contains one line per special file.

The first character of a line is either `0' or `1' or `2'; zero causes the line to be ignored, one causes it to be effective as a dialup line, and two specifies local terminal operation. The second character is used as an argument to getty (8), which performs such tasks as baud-rate recognition, reading the login name, and calling login. For normal lines, the character is `0'; other characters can be used, for example, with hard-wired terminals where speed recognition is unnecessary or which have special characteristics. (Getty will have to be fixed in such cases.) The remainder of the line is the terminal's entry in the device directory, /dev.

FILES

/etc/ttys

SEE ALSO

init(8), getty(8), login(1)

NAME

boot - startup procedures

DESCRIPTION

A PDP11 UNIX system is started by a two-stage process. The first is a primary bootstrap which is able to read in relatively small stand-alone programs; the second (called boot) is used to read in the system itself.

The primary bootstrap must reside in the otherwise unused block zero of the boot device. It can be read in and started by the standard ROM programs, or if necessary by keying in a small startup routine. This program is capable of loading type 407 executable files (not shared, not separate I&D). The user types on the system console the name of the program wished, in this case boot , followed by a carriage return; the named program is retrieved from the file system that starts at block 0 of drive 0 of the boot device. Successful loading of the primary bootstrap is signaled by the `#' prompt on the console, no diagnostic results if the file (/boot) cannot be found , and no provision is made for correcting typographical errors. In case the `/boot' file does not function, the `/boot.bu' file is provided as a backup copy of the bootstrap.

The second step, called boot, actually brings in the system. When read into location 0 and executed, boot sets up memory management, relocates itself into high memory, and types

```
Boot
:
```

on the console. Then it reads from the console a device specification (see below) followed immediately by a pathname. Boot finds the corresponding file on the given device, loads that file into memory location zero, sets up memory management as required, and calls the program by executing a `trap' instruction. Normal line editing characters can be used.

Conventionally, the name of the secondary boot program is `/boot' and the name of the current version of the system is `/unix'. Then, the recipe is:

1) Load block 0 of the boot device by fiddling with the console keys as appropriate for your hardware. If you have no appropriate ROM, some programs suitable for manual use are given below.

2) Respond to the `#' prompt by typing `boot' .

3) When the `boot' prompt is given, type one of the following:

```
hp(0,0)unix
hm(0,0)unix
hk(0,0)unix
rp(0,0)unix
rl(0,0)unix
```

depending on the type of disk you are loading from. The first 0 indicates the physical unit number; the second indicates the block number of the beginning of the logical file system to be searched. (See below).

When the system is running, it types

```
unix/v7m 2.0

mem = ######
#
```

on the console. The `mem =' message gives the amount of free memory in bytes, available to user programs. After setting the date (date(8)) and doing any file system checks a multi-user system is brought up by typing an EOT (control-d) in response to the `#' prompt.

Device specifications. A device specification has the following form:

```
device(unit,offset)
```

where device is the type of the device to be searched, unit is the unit number of the device, and offset is the block offset of the file system on the device. Device is one of the following

hp	RM02/3 or RP04/5/6 on first RH11/70
hm	RM02/3 or RP04/5/6 on second RH11/70
hk	RK06/7
rp	RP03
rl	RL01/2
rk	RK05

For example, the specification

```
rp(1,7000)
```

indicates an RP03 disk, unit 1, and the file system found starting at block 7000 (cylinder 35).

The `hp' disks have the ability to deal with interleaved file systems (see hp(4)). Unit numbers 0 thru 3 refer to physical drives 0 thru 3 non-interleaved, and unit numbers 4 thru 7 refer to physical drives 0 thru 3 with interleaved file systems.

ROM programs. The following programs to call the primary bootstrap may be installed in read-only memories or manually keyed into main memory. Each program is position-independent but should be placed well above location 0 so it will not be overwritten. Each reads a block from the beginning of a device into core location zero. The octal words constituting the program are listed on the left.

RK (drive 0):

```
012700      mov     $rkda,r0
177412
005040      clr     -(r0)           / rkda cleared by start
010040      mov     r0,-(r0)
012740      mov     $5,-(r0)
000005
105710  1:  tstb    (r0)
002376      bge     1b
005007      clr     pc
```

RP (drive 0)

```
012700      mov     $rpmr,r0
176726
005040      clr     -(r0)
005040      clr     -(r0)
005040      clr     -(r0)
010040      mov     r0,-(r0)
012740      mov     $5,-(r0)
000005
105710  1:  tstb    (r0)
002376      bge     1b
005007      clr     pc
```

FILES

/unix - system code
/mdec/hpuboot
/mdec/hkuboot
/mdec/rpuboot
/mdec/rluboot - copies of the prinmary bootstraps
/boot - second stage bootstrap
/boot.bu - backup copy of boot

SEE ALSO

init(8)

NAME

crash - what to do when the system crashes

DESCRIPTION

This section gives at least a few clues about how to proceed if the system crashes. It can't pretend to be complete.

Bringing it back up. If the reason for the crash is not evident (see below for guidance on `evident') you may want to try to dump the system if you feel up to debugging. At the moment a dump can be taken only on magtape. With a tape mounted and ready, stop the machine, load address 1000, and start. This should write a copy of all of core on the tape with an EOF mark. Caution: Any error is taken to mean the end of core has been reached. This means that you must be sure the ring is in, the tape is ready, and the tape is clean and new. The system will halt after the completion of a successful dump or hang in a tight loop if the dump fails. If the dump fails, you can try again, by loading address 1000 and restarting. The registers are saved on the first dump attempt only, so nothing is lost by restarting. See below for what to do with the tape.

In restarting after a crash, always bring up the system single-user. This is accomplished by following the directions in boot (8) as modified for your particular installation. When it is running, perform a fsck (1m) on all file systems which could have been in use at the time of the crash. If any serious file system problems are found, they should be repaired. When you are satisfied with the health of your disks, check and set the date if necessary, then come up multi-user. This is accomplished by typing an EOT (control d).

To even boot UNIX at all, three files (and the directories leading to them) must be intact. First, the initialization program /etc/init must be present and executable. If it is not, the CPU will loop in user mode at location 6. For init to work correctly, /dev/console and /bin/sh must be present. If either does not exist, the symptom is best described as thrashing. Init will go into a fork/exec loop trying to create a Shell with proper standard input and output.

If you cannot get the system to boot, a runnable system must be obtained from a backup medium. The root file system may then be doctored as a mounted file system as described below. If there are any problems with the root file system, it is probably prudent to go to a backup system to avoid working on a mounted file system.

Repairing disks. The first rule to keep in mind is that an addled disk should be treated gently; it shouldn't be

mounted unless necessary, and if it is very valuable yet in quite bad shape, perhaps it should be dumped before trying surgery on it. This is an area where experience and informed courage count for much.

The problems reported by icheck typically fall into two kinds. There can be problems with the free list: duplicates in the free list, or free blocks also in files. These can be cured easily with an icheck -s. If the same block appears in more than one file or if a file contains bad blocks, the files should be deleted, and the free list reconstructed. The best way to delete such a file is to use clri (1), then remove its directory entries. If any of the affected files is really precious, you can try to copy it to another device first.

Dcheck may report files which have more directory entries than links. Such situations are potentially dangerous; clri discusses a special case of the problem. All the directory entries for the file should be removed. If on the other hand there are more links than directory entries, there is no danger of spreading infection, but merely some disk space that is lost for use. It is sufficient to copy the file (if it has any entries and is useful) then use clri on its inode and remove any directory entries that do exist.

Finally, there may be inodes reported by dcheck that have Ø links and Ø entries. These occur on the root device when the system is stopped with pipes open, and on other file systems when the system stops with files that have been deleted while still open. A clri will free the inode, and an icheck -s will recover any missing blocks.

The icheck , dcheck and clri information given above is for reference only, fsck (1m) should be used instead of icheck and dcheck for file system repairs, see fsck (1m) for more detail.

Why did it crash? UNIX types a message on the console typewriter when it voluntarily crashes. Here is the current list of such messages, with enough information to provide a hope at least of the remedy. The message has the form `panic: ...', possibly accompanied by other information. Left unstated in all cases is the possibility that hardware or software error produced the message in some unexpected way.

blkdev
: The getblk routine was called with a nonexistent major device as argument. Definitely hardware or software error.

devtab
Null device table entry for the major device used as argument to getblk. Definitely hardware or software error.

iinit
An I/O error reading the super-block for the root file system during initialization.

no fs
A device has disappeared from the mounted-device table. Definitely hardware or software error.

no imt
Like `no fs', but produced elsewhere.

no clock
During initialization, neither the line nor programmable clock was found to exist.

I/O err in swap
An unrecoverable I/O error during a swap. Really shouldn't be a panic, but it is hard to fix.

out of swap space
A program needs to be swapped out, and there is no more swap space. It has to be increased. This really shouldn't be a panic, but there is no easy fix.

out of swap
No room in the swap area to hold the argument list while a process does an exec (2). Swap area size should be increased.

no procs
The process table has overflowed, increase NPROC in param.h.

timeout table overflow
The timeout table is not large enough.

zero wchan or sleeping on wchan Ø
A process is waiting on an address (wchan) of zero, which is illegal.

Running a dead proc
The system attempted to activate a "zombie" process.

parity
A memory parity error has occurred, can be accompanied by memory error registers on certian processors.

trap

An unexpected trap has occurred within the system. This is accompanied by five numbers: a `ka6', which is the contents of the segmentation register for the area in which the system's stack is kept; `aps', which is the location where the hardware stored the program status word during the trap; `pc', program counter and `ps', processor status word at the time of the trap; and a `trap type' which encodes which trap occurred. The trap types are:

0	bus error
1	reserved instruction
2	BPT/trace
3	IOT
4	power fail
5	EMT
6	recursive system call (TRAP instruction)
7	Programmed interrupt request
10	floating point trap
11	segmentation violation

In some of these cases it is possible for octal 20 to be added into the trap type; this indicates that the processor was in user mode when the trap occurred. If you wish to examine the stack after such a trap, either dump the system, or use the console switches to examine core; the required address mapping is described below.

Interpreting dumps. All file system problems should be taken care of before attempting to look at dumps. The dump should be read into the file /usr/sys/core; cp (1) will do. At this point, you should execute ps -alxk and who to print the process table and the users who were on at the time of the crash. You should dump (od (1)) the first 30 bytes of /usr/sys/core. Starting at location 4, the registers R0, R1, R2, R3, R4, R5, SP and KDSA6 (KISA6 for non-separate I & D CPU's) are stored. These are not the values of the registers at the time of the crash, those values are stored on the kernal stack, pointed to by `aps'. Next, take the value of KA6 (location 022(8) in the dump) multiplied by 0100(8) and dump 01000(8) bytes starting from there. This is the per-process data associated with the process running at the time of the crash. Relabel the addresses 140000 to 141776. R5 is C's frame or display pointer. Stored at (R5) is the old R5 pointing to the previous stack frame. At (R5)+2 is the saved PC of the calling procedure. Trace this calling chain until you obtain an R5 value of 141756, which is where the user's R5 is stored. If the chain is broken, you have to look for a plausible R5, PC pair and continue from there. Each PC should be looked up in the system's name list using adb (1) and its `:' command, to get a reverse calling order.

In most cases this procedure will give an idea of what is wrong. A more complete discussion of system debugging is impossible here.

SEE ALSO
clri(1), icheck(1), dcheck(1), boot(8), fsck(1m)

NAME

dmesg - collect system diagnostic messages to form error log

SYNOPSIS

/etc/dmesg [-]

DESCRIPTION

Dmesg looks in a system buffer for recently printed diagnostic messages and prints them on the standard output. The messages are those printed by the system when device (hardware) errors occur and (occasionally) when system tables overflow non-fatally. If the - flag is given, then dmesg computes (incrementally) the new messages since the last time it was run and places these on the standard output. This is typically used with cron (8) to produce the error log /usr/adm/messages by running the command

```
/etc/dmesg - >> /usr/adm/messages
```

every 10 minutes.

FILES

/usr/adm/messages	error log (conventional location)
/usr/adm/msgbuf	scratch file for memory of - option

BUGS

The system error message buffer is of small finite size. As dmesg is run only every few minutes, not all error messages are guaranteed to be logged. This can be construed as a blessing rather than a curse.

Error diagnostics generated immediately before a system crash will never get logged.

NAME

init, rc - process control initialization

SYNOPSIS

/etc/init
/etc/rc

DESCRIPTION

Init is invoked as the last step of the boot procedure (see boot (8)). Generally its role is to create a process for each typewriter on which a user may log in.

When init first is executed the console typewriter /dev/console. is opened for reading and writing and the shell is invoked immediately. This feature is used to bring up a single-user system. If the shell terminates, init comes up multi-user and the process described below is started.

When init comes up multiuser, it invokes a shell, with input taken from the file /etc/rc. This command file performs housekeeping like removing temporary files, mounting file systems, and starting daemons.

Then init reads the file /etc/ttys and forks several times to create a process for each typewriter specified in the file. Each of these processes opens the appropriate typewriter for reading and writing. These channels thus receive file descriptors Ø, 1 and 2, the standard input, output and error files. Opening the typewriter will usually involve a delay, since the open is not completed until someone is dialed up and carrier established on the channel. Local typewriters are opened immediately, see ttys(5). Then /etc/getty is called with argument as specified by the last character of the ttys file line. Getty reads the user's name and invokes login (1) to log in the user and execute the shell.

Ultimately the shell will terminate because of an end-of-file either typed explicitly or generated as a result of hanging up. The main path of init , which has been waiting for such an event, wakes up and removes the appropriate entry from the file utmp , which records current users, and makes an entry in /usr/adm/wtmp , which maintains a history of logins and logouts. Then the appropriate typewriter is reopened and getty is reinvoked.

Init catches the interrupt signal SIGINT and interprets it to mean that the system should be brought from multi user to single user. Use `kill -2 1' to send the interrupt signal. Init also catches the hangup signal SIGHUP, which causes init to reread the /etc/ttys file. To bring new terninals

on-line or take existing terminals off-line, edit the /etc/ttys file and use `kill -1 1' to send the hangup signal to init. Only the terminals whose flag character, in the /etc/ttys file, has been changed will be affected.

FILES

/dev/tty?, /etc/utmp, /usr/adm/wtmp, /etc/ttys, /etc/rc

SEE ALSO

login(1), kill(1), sh(1), ttys(5), getty(8)

NAME

shutdown - orderly system shutdown

SYNOPSIS

/etc/shutdown [time]

DESCRIPTION

The shutdown command allows the super-user and no one else to bring the system down in an orderly fashion. Shutdown may only be run from the console terminal. The optional time argument is the time delay in minutes until the shutdown will occur. If the time is not given shutdown will ask for the shutdown time delay.

Once invoked shutdown ; disables logins for all but the console terminal, sends a shutdown warning message to all logged in users every minute until shutdown time arrives, sends a final shutdown warning message, kills all processes except process group zero, dismounts all but the root file system, syncs the disks, and brings the system down to single-user mode. The shutdown may be aborted by typing a `delete' on the console terminal.

FILES

/etc/sdloglock	disables user logins
/unix	system namelist
/dev/mem	core memory
/dev	searched to find swap device and tty names

BUGS

Multi-user mode cannot be reestablished by typing control D (EOT) after the shutdown has reached single-user mode, the system must be rebooted.