Setting up the Fourth Berkeley Software Tape*
Draft of: November 15, 1980

William N. Joy
Ozalp Babaoglu
Keith Sklower

Computer Science Division
Department of Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, California 94720

The basic distribution tape can be used only on a DEC VAX-11/780** with RM03, RM05 or RP06 disks and with TE16, TU45 or TU77 tape drives. We have the ability to make tapes for systems with UNIBUS** disks, but such tapes are inherently rather system-specific, and will not be discussed here. The tape consists of some preliminary bootstrapping programs followed by one dump of a filesystem (see *dump*(8)††) and one tape archive image (see *tar*(1)); if needed, individual files can be extracted after the initial construction of the filesystems.

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

The tape contains binary images of the system and all the user level programs, along with source and manual sections for them. There are about 5600 UNIX† files altogether. The first tape file contains bootstrapping programs. The second tape file is to be put on one filesystem called the 'root filesystem', and contains essential binaries and enough other files to allow the system to run. The third tape file has all of the source and documentation. Altogether the files provided on the tape occupy approximately 52000 512 byte blocks.‡

Making a disk from tape

Before you begin to work on the remainder of this document, be sure you have an up to date manual, and that you have applied all updates to the manual which were provided with it, in the correct order.

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

1. Mount the magtape on drive 0 at load point, making sure that the ring is not inserted.
2. Mount a disk pack on drive 0. It is preferable that the pack be formatted (e.g. using the standard DEC standalone utility); we provide programs for formatting RM03's and RP06's below, but RM05's must be pre-formatted by the DEC utility.
3. Key in at location 50000 and execute the following boot program: You may enter in lower-case, the LSI-11 will echo in upper-case. The machine's printouts are shown in boldface, explanatory

---

*Portions of this document are adapted from "Setting Up Unix/32V Version 1.0" by Thomas B. London and John F. Reiser.

** DEC, VAX, UNIBUS and MASSBUS are trademarks of Digital Equipment Corporation.

† UNIX is a Trademark of Bell Laboratories.

††References of the form X(Y) mean the subsection named X in section Y of the UNIX programmer's manual.

‡UNIX traditionally talks in terms of 512 character blocks, and for consistency across different versions of UNIX and to avoid mass confusion, user programs in the Virtual Vax version of the system also talk in terms of 512 blocks, despite the fact that the file system allocates 1024 byte blocks of disk space. All user progras such as *ls*(1) and *du*(1) speak in terms of 512 byte blocks; only system maintenance programs such as *mkfs*(8), *fsck*(8) *dump*(8), and *df*(1), speak to 1024 byte blocks. It is true that i/o is most efficient in 1024 byte quantities, but it is most natural for the user to think of this as "2 blocks at a time." In any case, packs remain sectored 512 bytes per sector, and at the lowest driver levels the system deals with 512 byte disk records.

comments are within ( ). Terminate each line you type by carriage return or line-feed.

```
>>> HALT
>>> UNJAM
>>> INIT
>>> D 50000 20009FDE
>>> D+  D0512001
>>> D+  3204A101
>>> D+  C114C08F
>>> D+  A1D40424
>>> D+  008FD00C
>>> D+  C1800000
>>> D+  8F320800
>>> D+  10A1FE00
>>> D+  00C139D0
>>> D+  00000004
>>> E 50000/NE:A
...                          (machine prints out values, check typing)
>>> START 50000
```

The tape should move and the CPU should halt at location 5002A. If it doesn't, you probably have entered the program incorrectly. Start over and check your typing.

4. Start the CPU with

```
>>> START 0
```

5. The console should type

```
=
```

If you have an RM-05 you must already have a formatted pack, and should skip to step 7. If the disk pack is otherwise already formatted, skip to step 6. Otherwise, format the pack with:

```
(bring in standalone RP06 formatter)
=rp6fmt
format : Format RP06/RM03 Disk

MBA no. : 0          (format spindle on mba unit : 0          (format unit zero)
(this procedure should take about 20 minutes for an RP06, 10 for an RM03)
(some diagnostic messages may appear here)

unit : -1            (exit from formatter)
=                    (back at tape boot level)
```

6. Next, verify the readability of the pack via

(bring in RP06 verifier)
= rpread
dread : Read RP06/RM03 Disk

disk unit : 0 (specify unit zero)
start block : 0 (start at block zero)
no. blocks : (default is entire pack)

(this procedure should take about 10 minutes for a RP06)
(some diagnostic messages may appear here)
 # Data Check errors : nn (number of soft errors)
 # Other errors : xx (number of hard errors)
disk unit: −1 (exit from rpread)
= (back to tape boot)

If the number of ‘Other errors’ is not zero, consideration should be given to obtaining a clean pack before proceeding further.

7. Create the root file system with the following procedure:

(bring in a standalone version of the *mkfs* (8) program)
= mkfs
file sys size: 7942 (number of 1024 byte blocks in root)
file system: hp(0,0) (root is on drive zero; first filsys there)
isize = 5072 (number of inodes in root filesystem)
m/n = 3 500 (interleave parameters)
= (back at tape boot level)

You now have a empty UNIX root filesystem. To restore the data which you need to boot the system, type

(bring in a standalone *restor* (8) program)
= restor
Tape? ht(0,1) (unit 0, second tape file)
Disk? hp(0,0) (into root file system)
Last chance before scribbling on disk. (just hit return)
(30 second pause then tape should move)
(tape moves for a few minutes)
End of tape
= (back at tape boot level)

Now, you are ready to boot up

Booting UNIX

Now boot UNIX:

(load bootstrap program)
= boot
Boot
: hp(0,0)vmunix (bring in *vmunix* off root system)

The bootstrap should then print out the sizes of the different parts of the system (text, initialized and uninitialized data) and then the system should start with a message which looks (like):

87844+15464+130300 start 0x530
VM/UNIX (Berkeley Version 4.1) 11/10/80
real mem = *xxx*
avail mem = *yyy*
WARNING: preposterous time in file system −− CHECK AND RESET THE DATE!
ERASE IS CONTROL-H!!!
#

The *mem* messages give the amount of real (physical) memory and the memory available to user programs in bytes. For example, if your machine has only 512K bytes of memory, then xxx will be 524228, i.e. exactly 512K. The ‘‘ERASE-IS’’ message is part of /.profile which was executed by the root shell when it started. You will probably want to change /.profile somewhat.

UNIX is now running, and the ‘UNIX Programmer’s manual’ applies. The ‘#’ is the prompt from the Shell, and indicates you are the super-user. You should first check the integrity of the root file system by giving the command

fsck /dev/rrp0a

The output from *fsck* should look something like:

/dev/rrp0a
File System: Volume:

**** Checking /dev/rrp0a**
**** Phase 1 - Check Blocks and Sizes**
**** Phase 2 - Check Pathnames**
**** Phase 3 - Check Connectivity**
**** Phase 4 - Check Reference Counts**
**** Phase 5 - Check Free List**
282 files 1766 blocks 5677 free

If there are inconsistencies in the file system, you may be prompted as to whether to apply corrective action; see the document describing *fsck* for information.

Extracting /usr.

The next thing to do is to extract the rest of the data from the tape. Comments are enclosed in ( ); don’t type these. The number in the first command is the size of the filesystem to be created, in 1024 character blocks, just as given to the standalone version of *mkfs* above. (If you have an RM-03 rather than an RM-05 or RP-06 use ‘‘41040’’ rather than ‘‘145673’’ in the procedure below.)

```
# date yymmddhhmm              (set date, see date (1))
# passwd root                  (set password for super-user)
New password:                  (password will not echo)
Retype new password:
# /etc/mkfs /dev/rrp0g 145673  (create empty user filesystem)
isize = 65488                  (this is the number of available inodes)
m/n = 3 500                    (freelist interleave parameters)
(this takes a few minutes)
# /etc/mount /dev/rp0g /usr    (mount the usr filesystem)
# cd /usr                      (make /usr the current directory)
# cp /dev/rmt12 /dev/null      (skip first tape file (tp format))
# cp /dev/rmt12 /dev/null      (skip second tape file (root))
# tar xpb 20                   (extract the usr filesystem)
(this takes about 20 minutes)
# rmdir lost+found
# /etc/mklost+found            (a directory for fsck)
# dd if=/usr/mdec/uboot of=/dev/rrp0a bs=1b count=1
(write boot block so reboot(8) disk boot scheme will work)
# cd /                         (back to root)
# chmod 755  /  /usr
# /etc/umount /dev/rp0g        (unmount /usr)
```

All of the data on the tape has now been extracted. The tape will rewind automatically.

You should now check the consistency of the /usr file system by doing

```
# fsck /dev/rrp0g
```

In order to use the /usr file system, you should now remount it by saying

```
# /etc/mount /dev/rp0g /usr
```

Making a UNIX boot floppy

The next thing to do is to make a new UNIX boot floppy, by adding some files to a copy of your current console floppy, using *flcopy* and *arff*(8).

Place your current floppy in the console, and issue the following commands:

```
# cd /usr/src/sys/floppy
# mkdir fromdec                (scratch sub-directory)
# cd fromdec
# arff xv                      (extract all files from floppy)
(list of files prints out)
# flcopy −t3
(system reads header information off current floppy)
Change Floppy, Hit return when done.
(waits for you to put clean floppy in console)
(copies header information back out after you hit return)
# rm floppy                    (don't need copy of old header information)
# rm dm* db* vmb.exe           (don't need these files with UNIX)
# arff cr *                    (add basic files)
Are you sure you want to clobber the floppy?
yes                            (clobbering is, essentially, a mkfs)
# cd ..
# rm −r fromdec                (remove scratch directory)
# arff r *                     (add UNIX boot files to floppy)
```

More copies of this floppy can be made using **flcopy.**

You should now be able to reboot using the procedures in *reboot*(8). First you should turn on the auto-reboot switch on the machine. Then try a reboot, saying

```
# /etc/reboot −s
```

which you can read about in *reboot*(8).

Taking the system up and down

Normally the system reboots itself and no intervention is needed at the console command level (e.g. to the LSI-11 ‘‘>>>’’ prompt.)† In fact, after such a reboot, the system normally performs a reboot checking the disks, and then goes back to multi-user mode. Such a reboot can be stopped (after it prints the date) with a delete (interrupt).

If booting from the console command level is needed, then the command

```
>>> B RPS
```

will boot from unit 0 on mba 0 (RM-03, RM-05 or RP-06), bringing in the file ‘‘hp(0,0)vmunix’’. Other possibilities are ‘‘B RPM’’ which boots and runs the automatic reboot procedure or ‘‘B ANY’’ which boots and asks for the name of the file to be booted. Note that ‘‘B’’ with no arguments is used internally by the reboot code, and shouldn’t be used.

To bring the system up to a multi-user configuration from the single-user status after, e.g., a ‘‘B RPS’’ all you have to do is hit control-d on the console. The system will then perform /etc/rc, a multi-user restart script, and come up on the terminals which are indicated in /etc/ttys. See *init*(8) and *ttys*(5). Note, however, that this does not cause a file system check to be performed. Unless the system was taken down cleanly, you should run ‘‘fsck −p’’ or force a reboot with *reboot*(8) to have the disks checked.

To take the system down to a single user state you can use

```
# kill 1
```

when you are up multi-user. This will kill all processes and give you a shell on the console, as if you had just booted.

If you wish to change the terminal lines which are active you can edit the file /etc/ttys, changing the first characters of lines, and then do

```
# kill −1 1
```

See *init*(8) and *getty*(8) for more information.

Backing up the system

Before you change the source for the kernel it is wise to make a backup. The following will do this:

```
# cd /usr/src
# mkdir distsys distsys/h distsys/sys distsys/dev distsys/conf distsys/stand
# cd sys/sys
# cp * /usr/src/distsys/sys
# cd ../h
# cp * /usr/src/distsys/h
# cd ../dev
# cp * /usr/src/distsys/dev
# cd ../conf
# cp * /usr/src/distsys/conf
# cd ../stand
# cp * /usr/src/distsys/stand
```

This allows you to find out what you have done to the distribution system by later running the command

†If you are going to make your root device be something other than a MASSBUS disk, you will have to change the file RESTAR.CMD on the floppy, to pass the block device index of the major device to be booted from to the system in a register after, e.g., a power-fail.

diff(1), comparing these directories.

Organization

The system source is kept in the subdirectories of /usr/src/sys. The directory **sys** contains the mainline kernel code, implementing system calls, the file system, virtual memory, etc. The directory **dev** contains device drivers and other low-level routines. The directory **conf** contains CPU-dependent information on physical and logical device configuration. The directory **h** contains all the header files, defining structures and system constants. **N.B.: The system header files in /usr/src/sys/h are copies of the files in /usr/include/sys. Since programs which depend on constants in /usr/include/sys/param.h must correspond to the running system, you should be careful to make new header files available whenever you resize the system or otherwise change the header files.**

You should expect to have to make some changes to the **conf** directory and perhaps to some of the header files in the **h** directory to resize some things.

The conf, sys and dev directories each have their own *makefile* controlling recompilation. The **sys** makefile is the master makefile; new versions of the kernel are created as ‘‘vmunix’’ in the **sys** directory. If changes are made in other directories, then the following commands:

```
# cd ../sys
# rm vmunix
# make
```

will cause all needed recompilations to be done. It is often convenient when making changes in **dev** or **conf** to just run *make* there first.

Finally note that the directory **stand** which is not part of the system proper. If you add new peripherals such as tapes or disks you will want to extend the standalone system to be able to deal with these. It too, has a *makefile* which you can look at.

Isolating local changes conditionally

You will notice that the system, as distributed, has conditional code in it. The current Berkeley system, on ‘‘Ernie Co-vax’’ is made by defining IDENT in the **makefile**s to be

```
IDENT= −DERNIE −DUCB
```

This enables the conditional code both for Berkeley and for this particular machine. It is traditional to pick a monicker for your machine, and change IDENT to reflect it, and to then put in changes conditionally whenever this makes sense. You can be guided by the ERNIE conditional code, which you will probably want to disable.

Device drivers

The UNIX system running is configured to run with the given disk and tape, a console, 32 DZ11 lines, 32 DH11 lines, a Varian printer/plotter, a Versatec printer/plotter, and 2 AMPEX 9300 disks on an EMULEX controller on the UNIBUS. This is probably not the correct configuration. It is easy to correct the configuration information to reflect the true state of your machine. For each device driver there there are certain magic numbers and configuration parameters kept in header files (suffixed **.h**) in the **conf** directory that you can change. The device addresses of each device are defined there, as are other specifications such as the number of devices. You should edit each **.h** file in **conf** change them as appropriate for your machine.

If you have any non-standard device addresses, just change the address and recompile. If the devices’s interrupt vector address(es) are different from those currently known to the system (this is likely), then the file /usr/src/sys/conf/univec.c must be modified appropriately: namely, the proper interrupt routine addresses must be placed in the table ‘UNIvec’. Now, make sure you add any new drivers which you have to the list of DRIVERS in the **makefile** in **dev**, and to the FILES and CFILES variables in this **makefile** and FILES3 and CFILES3 in **../sys/makefile**.

As describe in ‘‘Basics of disk layout’’ below, the first line of the makefile in the **sys** directory controls the root device and swap disk layout by picking one of the confhp.c, confup.c, etc. files of the **conf** directory. This should be chosed to reflect the desired disk layout before the system is recompiled.

Non-integrated device drivers

As of this date, several device drivers which were available at various times have not been integrated into the conditional compilation of the **dev** and **conf** directories. The source for these device drivers is in the directory **/usr/src/sys/newdev.** These device drivers will be made part of **/usr/src/sys/dev** and **/usr/src/sys/conf** as we have opportunity to test them. It should not be hard for you to integrate these drivers; if you wish, you can contact us as we will be able to supply integrated versions very soon. Here is a list of the drivers which are currently awaiting integration:

dhdm	DM11 modem control driver
kl	KL/DL-11 communications driver
lp	LP11 line printer driver
rk7	RK07 driver
tm	TM11 unibus tape driver

In addition, there is another rk07 driver (hk.c). Code to allow mixing of tapes and disks among and across several MBA’s is also in the works, but is not supplied on this tape. Contact us if you have need for this code.

If you integrate these or other drivers we would appreciate receiving a copy in the mail so that we can avoid duplication of effort. (Other comments on the system or these instructions are, of course, also welcomed.)

Makefile entry points

The *makefile* s have several additional useful entry points:

clean	Cleans out the directory, removing **.o** files and the like.
lint	(In **sys**) Runs lint on the system.
depend	Creates a new makefile indicating dependencies on header files by running a search through **.c** files looking for ‘‘#include’’ lines. Make sure you format your code like the rest of the system so that this will work.
print	(In **sys**) Produces a nice listing of most everything in the system directory in a canonical order.
symbols.sort	(In **sys**) Creates a new file for sorting symbols in the system namelist. If you have locally written programs which use the system namelist you can put the symbols which they reference in *symbols.raw* and they will be moved at system generation to the front of the system namelist for quicker access.
tags	Creates a *tags* file for *ex,* to make editing of the system much easier.

Basic constants

Before running *make,* you should check the definition of the constants in /usr/src/sys/h/param.h The constants NBUF, NINODE, NFILE, NPROC, and NTEXT can be changed, and also TIMEZONE and perhaps HZ if you run on 50 cycles.† There are also tunable constants in the file /usr/src/sys/h/vmtune.h but ignore them for the time being. As distributed, the system is tuned for a fairly large machine (i.e. 2+ Megabytes of memory and 2-3 disk arms).

To generate a completely recompiled UNIX do

† If you change NINODE, NFILE, NPROC or NTEXT, then the programs *analyze*(1), *ps*(1), *pstat*(1) and *w*(1) will have to be recompiled. A procedure for doing this is given below.

```
# cd /usr/src/sys/sys
# make clean ; cd ../dev ; make clean ; cd ../conf ; make clean
# cd ../sys
# make depend ; cd ../dev ; make depend ; cd ../conf ; make depend
# cd ../sys
# make −k > ERRS 2>& 1 &
(compilation will finish about 15 minutes later leaving output in ERRS)
```

The final object file (vmunix) should be moved to the root, and then booted to try it out. It is best to name it /newvmunix so as not to destroy the working system until you're sure it does work. It is also a good idea to keep the old working version around under some other name. A systematic scheme for numbering and saving old versions of the system is best. **Be sure to always have the current system in /vmunix when you are running multi-user or commands such as** *ps* (1) **and** *w* (1) **will not work.**

Special Files

Next you should remove any unnecessary device entries from the directory **/dev,** as devices were provided for all the things initially configured into the system. See how the devices were made using **MAKE** in the directory **/dev,** and make another directory **/newdev,** copy **MAKE** into it, edit **MAKE** to provide an entry for local needs, and rerun it to generate a **/newdev** directory. You can then do

```
# cd /
# mv dev olddev ; mv newdev dev
```

If you prefer, you can just whittle away at **/dev** using **rm, mv** and *mknod*(8) to make what you need, but if you do this stuff manually you may have to do it manually again someday, and the devices will appear much more ''magic'' to those who follow you.

Notes on the configuration file and device names

Print the configuration file /usr/src/sys/conf/conf.c. 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. 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 devices.

Each device is typically given several special files in **/dev.** The name is made from the name of the device driver, sometimes varying for historical reasons. Thus the ''up'' disk driver has disks ''up0a'', ''up0b'', etc., the partitions of drive 0 being given letters a–h.

The tape drives are given names not dependent on the tape driver hardware, since they are used directly by a number of programs. The file **mt0** is a 1024 byte blocked tape drive at 800 bpi; **mt8** is 1600 bpi. By adding 4 to the unit number you get non-rewinding tapes.

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 conventionally has the name of the corresponding standard block special file with 'r' prepended. Thus the raw magtape files are called /dev/rmtX.

The names for terminals should be named /dev/ttyX, where X is some string (as in '0' or 'd0'). While it is possible to use truly arbitrary strings here, the accounting and noticeably the *ps*(1) command make good use of the fact that tty names (at Berkeley) are distinct in the last 2 characters. In fact, we use the following convention: ''ttyN'', with N the minor device number for normal DZ ports; ''ttydX'' with X a single hex digit (starting from 0) for dialups, ''ttyhX'' and ''ttyiX'' with X a hex digit for dh ports, and ''console'' (abbrev ''co'') for the console. This works out well and *ps*(1) uses a heuristic algorithm based on these conventions to speed up name determination from device numbers it otherwise obtains.

Whenever special files are created, care should be taken to change the access modes *(chmod*(8)) on these files to appropriate values.

Basics of Disk Layout

If there are to be more filesystems mounted than just the root and /usr, use *mkfs*(8) and *mklost+found*(8) to create any new filesystem and put the information about it into the file **/etc/fstab** (see *fstab*(5)). You should look also at /etc/rc to see what commands are executed there and investigate how **fstab** is used.

Each physical disk drive can be divided into upto 8 partitions; we typically use only 3 partitions (or 4 on 300M drives). The first partition, e.g. **rp0a** is used for a root file system, a backup thereof, or a small file system like **/tmp.** The second partition **rp0b** is used for paging and swapping. The third partition **rp0g** is used to hold a user file system. We partition our 300M disks so that almost all of a large disk will fit onto a 200M disk, just dropping the last partition. This makes it easier to deal with hardware failures by just copying data.

There are several 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. Paging space is an important parameter. The system as distributed has 33440 (512 byte) blocks in which to page on the primary device; additional devices may be provided, with the paging interleaved between them. This is controlled by entries in **/etc/fstab** and in a configuration file in the **conf** directory.

The first line of the **makefile** in the **sys** directory selects one a file such as **confrp.c** in the conf directory. This file specifies the root and pipe devices and the devices which are to be used for swapping and paging. Each device which is to be used for paging (except the primary one) should be specified as a ‘‘sw’’ device in **fstab.**

Many common system programs (C, the editor, the assembler etc.) create intermediate files in the /tmp directory, so the filesystem where this is stored also should be made large enough to accommodate most high-water marks; if you have several disks, it makes sense to mount this in one of the other ‘‘root’’ (i.e. first partition) file systems. The root filesystem as distributed is quite large, and 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*(8), threatening messages of the day, and personal letters.

The efficiency with which UNIX is able to use the CPU is often strongly affected by the configuration of disk controllers. For general time-sharing applications, the best strategy is to try to split the root filesystem (/), system binaries (/usr), the temporary files (/tmp), and the user files among several disk arms, and to interleave the paging activity among a number of arms. We will discuss such considerations more below.

Moving filesystem data

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 filesystem is to dump it to magtape using *dump*(8), to use *mkfs*(8) and *mklost+found*(8) to create the new filesystem, and restore (*restor* (8)) 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 filesystem has to be increased in logical size without copying. The superblock 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 *fsck*(8). 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 *filsys*(5) before playing around in this way.

If you have to merge a filesystem into another, existing one, the best bet is to use *tar*(1). If you must shrink a filesystem, the best bet is to dump the original and restor it onto the new filesystem. However, this will 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 filesystem and only have one drive the procedure is more complicated. What you do is the following:

1. GET A SECOND PACK!!!!
2. Dump the root filesystem to tape using *dump*(8).
3. Bring the system down and mount the new pack.
4. Load the standalone versions of *mkfs*(8) and *restor*(8) from the floppy with a procedure like:

```
>>>UNJAM
>>>INIT
>>>LOAD MKFS
        LOAD DONE, xxxx BYTES LOADED
>>>ST 2

...

>>>H
        HALTED AT yyyy
>>>U
>>>I
>>>LOAD RESTOR
        LOAD DONE, zzzz BYTES LOADED

... etc
```

5. Boot normally using the newly created disk filesystem.

Note that if you change the disk partition tables or add new disk drivers they should also be added to the standalone system in /usr/src/sys/stand.

System Identification

You should edit the files:

```
/usr/include/ident.h
/usr/include/whoami.h
/usr/include/whoami
/usr/src/cmd/uucp/uucp.h
```

to correspond to your system, and then recompile and install *getty*(8), *binmail*(1), *who*(1) and *uucp*(1) via:

```
# cd /usr/src/cmd
# DESTDIR=/
# export DESTDIR
# MAKE getty.c mail.c who.c uucp
```

This will arrange for an appropriate banner to be printed on terminals before users log in, and also arrange that *uucp* knows what site you are.

The program *delivermail*(8) needs to know the topology of your network configuration to be able to forward mail properly; if you are on any networks other than *uucp* be sure to print the file **/usr/src/cmd/delivermail/READ_ME,** edit the appropriate tables, and recompile and install delivermail as you did **uucp** above.

If you run the Berkeley network *net*(1) be sure to change the constant LOCAL in /usr/src/cmd/ucb-mail/v7.local.h and recompile when you bring up a network machine.

Adding New Users

See *adduser*(8); local needs will undoubtedly dictate a somewhat different procedure.

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). You should also edit the file /etc/ttytype placing the type of the new terminal there (see *ttytype* (5)). The file /etc/ttywhere is also a useful one to keep up to date.

Note that /usr/src/cmd/init.c and /usr/src/cmd/comsat.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 week or so in the absence of any problems) and always (usually automatically) after a crash, all the filesystems should be checked for consistency by *fsck*(1). The procedures of *reboot*(8) should be used to get the system to a state where a file system check can be performed manually or automatically.

Dumping of the filesystems 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*(8). You should arrange to do a towers-of-hanoi dump sequence; we tune ours so that almost all files are dumped on two tapes and kept for at least a week in most every case. We take full dumps every month (and keep these indefinitely). Operators can execute ‘‘dump w’’ at login which will tell them what needs to be dumped (based on the **fstab** information). Be sure to create a group **operator** in the file **/etc/group** so that dump can notify logged-in operators when it needs help.†

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 with *dd*(1) using the raw special files and an appropriate block size.

† More precisely, we have three sets of dump tapes: 10 daily tapes, 5 weekly sets of 2 tapes, and fresh sets of three tapes monthly. We do daily dumps circularly on the daily tapes with sequence ‘3 2 5 4 7 6 9 8 9 9 9 ...’. Each weekly is a level 1 and the daily dump sequence level restarts after each weekly dump. Full dumps are level 0 and the daily sequence restarts after each full dump also. Thus a typical dump sequence would be:

tape name	level number	date	opr	size
FULL	0	Nov 24, 1979	jkf	137K
D1	3	Nov 28, 1979	jkf	29K
D2	2	Nov 29, 1979	rrh	34K
D3	5	Nov 30, 1979	rrh	19K
D4	4	Dec 1, 1979	rrh	22K
W1	1	Dec 2, 1979	etc	40K
D5	3	Dec 4, 1979	rrh	15K
D6	2	Dec 5, 1979	jkf	25K
D7	5	Dec 6, 1979	jkf	15K
D8	4	Dec 7, 1979	rrh	19K
W2	1	Dec 9, 1979	etc	118K
D9	3	Dec 11, 1979	rrh	15K
D10	2	Dec 12, 1979	rrh	26K
D1	5	Dec 15, 1979	rrh	14K
W3	1	Dec 17, 1979	etc	71K
D2	3	Dec 18, 1979	etc	13K
FULL	0	Dec 22, 1979	etc	135K

We do weekly’s often enough that daily’s always fit on one tape and in fact never get to the sequence of 9’s in the daily level numbers.

Converting 32/V Filesystems

The best way to convert filesystems from 32/V to the new format is to use *tar*(1). After converting, you can still restore files from your old-format dump tapes (yes the dump format is different, sorry about that), by using ‘‘512restor’’ instead of ‘‘restor’’. If you wish, you can move whole file systems from 32/V to the new system by using ‘‘dump’’ and then ‘‘512restor’’.

Regenerating the system

It is quite easy to regenerate the system, and it is a good idea to try this once right away to build confidence. The system consists of three major parts: the kernel itself (/usr/src/sys/sys), the user programs (/usr/src/cmd and subdirectories), and the libraries (/usr/src/lib*). The major part of this is /usr/src/cmd.

We have already seen how to recompile the system itself. The three major libraries are the C library in /usr/src/libc and the FORTRAN libraries /usr/src/libI77 and /usr/src/libF77. In each case the library is remade by changing into the corresponding directory and doing

```
# make
```

and then installed by

```
# make install
```

Similar to the system,

```
# make clean
```

cleans up. The source for all other libraries is kept in subdirectories of /usr/src/lib; each has a makefile and can be recompiled by the above recipe.

Recompiling all user programs in /usr/src/cmd is accomplished by using the MAKE shell script which resides there and its associated file DESTINATIONS. For instance, to recompile ‘‘date.c’’, all one has to do is

```
# cd /usr/src/cmd
# MAKE date.c
```

this will place a stripped version of the binary of ‘‘date’’ in /4bsd/bin/date, since date normally resides in /bin, and Admin is building a file-system like tree rooted at /4bsd You will have to make the directory 4bsd for this to work. It is possible to use any directory for the destination, it isn't necessary to use the default /4bsd; just change the instance of ‘‘4bsd’’ at the front of MAKE.

You can also override the default target by doing:

DESTDIR=*pathname*
export DESTDIR

To regenerate all the system source you can do

```
# DESTDIR=/usr/newsys
# export DESTDIR
# cd /usr
# rm −r newsys
# mkdir newsys
# cd /usr/src/cmd
# MAKE * > ERRS 2>& 1 &
```

This will take about 4 hours on a reasonably configured machine. When it finished you can move the hierarchy into the normal places using *mv*(1) and *cp*(1), and then execute

```
# DESTDIR=/
# export DESTDIR
# cd /usr/src/cmd
# MAKE ALIASES
# MAKE MODES
```

to link files together as necessary and to set all the right set-user-id bits.

Making orderly changes

In order to keep track of changes to system source we migrate changed versions of commands in /usr/src/cmd in through the directory /usr/src/new and out of /usr/src/cmd into /usr/src/old for a time before removing them. Locally written commands which aren't distributed are kept in /usr/src/local and their binaries are kept in /usr/local. This allows /usr/bin /usr/ucb and /bin to correspond to the distribution tape (and to the manuals that people can buy). People wishing to use /usr/local commands are made aware that they aren't in the base manual. As manual updates incorporate these commands they are moved to /usr/ucb.

A directory /usr/junk to throw garbage into, as well as binary directories /usr/old and /usr/new are very useful. The man command supports manual directories such as /usr/man/manj for junk and /usr/man/manl for local to make this or something similar practical.

Interpreting system activity

The *vmstat* program provided with the system is designed to be an aid to monitoring systemwide activity. Together with the *ps*(1) command (as in ‘‘ps av’’), it can be used to investigate systemwide virtual activity. By running *vmstat* when the system is active you can judge the system activity in several dimensions: job distribution, virtual memory load, paging and swapping activity, disk and cpu utilization. Ideally, there should be few blocked (B) jobs, there should be little paging or swapping activity, there should be available bandwidth on the disk devices (most single arms peak out at about 30-35 tps in practice), and the user cpu utilization (US) should be high (above 60%).

If the system is busy, then the number of active jobs may be large, and several of these jobs may often be blocked (B). If the virtual memory is very active, then the paging demon may be running (SR will be non-zero). It is healthy for the paging demon to free pages when the virtual memory gets active; it is triggered by the amount of free memory dropping below a threshold and increases its pace as free memory goes to zero.

If you run *vmstat* when the system is busy (a ‘‘vmstat 5’’ is best, since that is how often most of the numbers are recomputed by the system), you can find imbalances by noting abnormal job distributions. If a large number of jobs are blocked (B), then the disk subsystem is overloaded or imbalanced. If you have a large number of non-dma devices or open teletype lines which are ‘‘ringing’’, or user programs which are doing high-speed non-buffered input/output, then the system time may go very high (60-70% or higher). It is often possible to pin down the cause of high system time by looking to see if there is excessive context switching (CS), interrupt activity (IN) or system call activity (SY). Cumulatively on one of our large machines we average about 60 context switches and interrupts per second and about 90 system calls per second.

If the system is very heavily loaded, or if you have very little memory relative to your load (1M is little in most any case), then the system may be forced to swap. This is likely to be accompanied by a noticeable reduction in system performance as the system does not swap ‘‘working sets’’, but rather forces jobs to reinitialize their resident sets by demand paging. If you expect to be in a memory-poor environment for an extended period you might consider administratively limiting system load, and should be sure to downsize the system.

Tunable constants

There is a modicum of tuning available in the virtual memrory management mechanism if it appears to be badly tuned for your configuration. The page replacement (clock) algorithm is run whenever there are not LOTSFREE pages available (this and all other constants discussed here are defined in the system header file /usr/src/sys/h/vmtune.h). This sets up resistance to consumption of the remaining free memory at a minimal rate SLOWSCAN, which gives the desired number of seconds between successive examinations of each page. The rate at which the clock algorithm is run increases linearly to a desired rate of FASTSCAN when there is no free memory. Thus as the available free memory decreases, the clock algorithm works harder to hold on to what is left. If less than DESFREE pages are available and the paging rate is high, then the system will begin to swap processes out. If less than MINFREE pages are available then the system will begin to swap, regardless of the paging rate.

When it has to swap, the system first tries to find a process which has been blocked for a long time and swap it out first. If there are no jobs of this flavor, then it will choose among the 4 largest jobs in-core which it can swap, picking the one of these which has been core resident longest. It attempts to guarantee (during periods of very heavy load) enough core residency to a process to allow it to at least rebuild its set of active pages (since it must do so by demand paging). Processes which are swapped out with large numbers of active pages similarly receive lower priority for swapin, favoring small jobs to return to the core resident set quickly.

It is *very* desirable that the system run under reasonably heavy load with little swapping, with the memory partitioning being done by the clock replacement algorithm, rather than by the swapping algorithm. The costs associated with paging activity are the time spent in the paging demon, the overhead associated with reclaim page faults (RE), and the extra disk activity associated with pagins and pageouts. We will discuss disk considerations later; when kept to about 40 reclaim faults per second, the cost of reclaims is less than 1% of total processor time. The cpu time (shown by ‘‘ps u2’’) accumulated by the pageout demon will show how much overhead it is generating.

The system, as distributed, runs the replacement algorithm whenever less than 1/4 of the total user memory is free. This is done starting with a 25 second revolution time of the clock algorithm and increasing to a 15 second revolution time when there is no free memory. The goal here is to use as much memory as possible (i.e. have the free list short) but to not have the system run out and start to swap. You can experiment with changing the writable copies of these variables (e.g. ‘‘lotsfree’’ is the writable copy of LOTSFREE) using *adb,* as in:

```
# adb −w /vmunix /dev/kmem
lotsfree/D
---adb prints value of lotsfree---
/W 0t100
```

Here the ‘‘/W 0t100’’ command changed the value of *lotsfree* to be 100 (decimal).

One final constant which can be changed is **klin** which controls the page-in clustering in the system. As distributed, it is set to 2, which allows the system to pre-page a even numbered (1k byte) page when an odd numbered page is faulted and vice-versa. In extreme circumstances, for special purpose applications which cause heavy paging activity you might try setting it to 4 to increase the amount of pre-paging, allowing the system to pre-page up to 3 adjacent pages. This will increase the rate at which memory is consumed on an active system, so you should be aware that this can overload the page replacement mechanisms ability to maintain enough free memory and can thus cause swapping. With this volume of page traffic it may be necessary to set the global constant **fifo** to 1 in the system, causing the paging algorithm to ignore page-referencing behavior and favoring, rather, circular replacement.

Klin should be changed only experimentally on systems with abnormal amounts of paging activity. This is not necessary or appropriate for most any normal timesharing load.

Balancing disk load

It is critical for good performance to balance disk load. There are at least five components of the disk load which you can divide between the available disks:

1. The root file system.
2. The /tmp file system.
3. The /usr file system.
4. The user files.
5. The paging activity.

The following possibilities are ones we have actually used at times when we had 2, 3 and 4 disks:

	disks		
what	2	3	4
/	1	2	2
tmp	1	3	4
usr	1	1	1
paging	1+2	1+3	1+3+4
users	2	2+3	2+3
archive	x	x	4

Splits such as these should get you going. The most important things to remember are to even out the disk load as much as possible, and to do this by decoupling file systems (on separate arms) between which heavy copying occurs. Note that a long term average balanced load is not important... it is much more important to have instantaneously balanced load when the system is busy.

Intelligent experimentation with a few file system arrangements can pay off in much improved performance. It is particularly easy to move the root, the /tmp file system and the paging areas. Place the user files and the /usr directory as space needs dictate and experiment with the other, more easily moved file systems.

Process size limitations

As distributed, the system provides for a maximum of 64M bytes of resident user virtual address space. The size of the text, and data segments of a single process are currently limited to 6M bytes each, and the stack segment size is limited to 512K bytes as a soft, user-changeable limit, and may be increased to 6M by calling *vlimit*(2). If these are insufficient, they can be increased by changing the constants MAXTSIZ, MAXDSIZ and MAXSSIZ in the file /usr/src/sys/h/vm.h, while changing the definitions in /usr/src/sys/h/dmap.h and /usr/src/sys/h/text.h. You must be careful in doing this that you have adequate paging space. As configured above, the system has only 16M bytes of paging area, since there is only one paging area. The best way to get more space is to provide multiple, thereby interleaved, paging areas by using a file other than confhp.c; see the first line of the makefile in the sys directory and the disk layout section above.

To increase the amount of resident virtual space possible, you can alter the constant USRPTSIZE (in /usr/src/sys/h/vm.h) and by correspondingly change the definitions of *Usrptmap* in /usr/src/sys/sys/locore.s Thus to allow 128 megabytes of resident virtual space one would declare

```
_Usrptmap: .space16*NBPG
```

The system has 6 pages of page tables for its text+data+bss areas and the per-page system information. This limits this part of the system to 6*64K = 384K bytes. The per-page system information uses 12 bytes per 1024 bytes of user available physical memory. This (conservatively) takes 55K bytes on a 4 megabyte machine, limiting the ‘‘size’’ of the system to about 330K bytes. You can increase the size of the system page table to, for example, 8 pages by defining, in locore.s:

```
_Sysmap:    .space8*NBPG
```

You will then also have to change the definitions of the constants UBA0, MBA0, and MBA1 in the files **uba.h mba.h, mba.m,** and **uba.m.** The 6 in the numbers here is the 6 from the number of pages in *Sysmap* thus UBA0 would then be defined by

#define UBA0 0x80080000

You should also change the constant RELOC in the bootstrap programs in **/usr/src/sys/stand** to be large enough to relocate past the end of the system. *Grep*(1) for RELOC in this directory and change the constants, recompile the bootstrap programs and replace them on the floppy.

Other limitations

Due to the fact that the file system block numbers are stored in page table **pg_blkno** entries, the maximum size of a file system is limited to 2^20 1024 byte blocks. Thus no file system can be larger than 1024M bytes.

The number of mountable file systems is limited to 15 (which should be enough; if you have a lot of disks it makes sense to make some of them single file systems, and the paging areas dont count in this total.) To increase this it will be necessary to change the core-map /usr/src/sys/h/cmap.h since there is a 4 bit field used here. The size of the core-map will then expand to 16 bytes per 1024 byte page and you should change /usr/src/sys/h/cmap.m also. (Don't forget to change MSWAPX and NMOUNT in /usr/src/sys/h/param.h also.)

The maximum value NOFILE (number of open files per process) can be raised to is 30 because of a bit field in the page table entry in **/usr/src/sys/h/pte.h.**

Scaling down

If you have 1.5M byte of memory or less you may wish to scale the paging system down, by reducing some fixed table sizes not directly related to the paging system. For instance, you could reduce NBUF from 128 to 40, NCLIST from 500 to 150, NPROC from 250 to 125, NINODE from 400 to 200, NFILE from 350 to 175, NMOUNT from 15 to 8, and NTEXT from 60 to 40. You can use *pstat*(8) with the **−T** option to find out how much of these structures are typically in use. Although the document is somewhat outdated for the VAX, you can see the last few pages of "Regenerating System Software" in Volume 2B of the programmers manual for hints on setting some of these constants.

Files which need attention

The following files require periodic attention or are system specific

/etc/fstab	how disk partitions are used
/etc/group	group memberships
/etc/motd	message of the day
/etc/passwd	password file; each account has a line
/etc/rc	system restart script; runs reboot; starts daemons
/etc/ttys	enables/disables ports
/etc/ttytype	terminal types corrected to ports
/etc/ttywhere	lists physical locations of terminals
/usr/lib/crontab	commands which are run periodically
/usr/lib/mailaliases	mail forwarding and distribution groups
/usr/adm/acct	raw process account data
/usr/adm/dnacct	raw autodialer account data
/usr/adm/messages	system error log
/usr/adm/wtmp	login session accounting

Thats all for now.

Good luck.

William N. Joy
Ozalp Babaoglu
Keith Sklower