Setting up the Third Berkeley Software Tape*

William N. Joy
Ozalp Babaoglu

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

The distribution tape can be used only a DEC VAX-11/780** with RM03 or RP06 disks and with TE16 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*(1)) 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 4200 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 40000 512 byte blocks.‡

Making a disk from tape

This description is an annotated version of the 'sysgen' manual page in section 8 of the UNIX Programmer's Manual. Before you begin to work on the remainder of this manual, 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.
3. Key in at 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 comments are within ( ). Terminate each line you type by carriage return or line-feed.

*An early version of this paper appeared under the title "Setting up the Berkeley Virtual Memory Extensions to the UNIX Operating System" and, no doubt, references to this paper by this name exist elsewhere in the documentation. 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.

‡ 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*(1), *icheck*(1), *dump*(1), 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.


```
>>> 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 the disk pack is 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)
   (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)
(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 (1) 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 (1) program)
=restor
Tape? ht(1,1)              (1600 bpi, 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)
...
=                      (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):


```
61072+61080+70120 start 0x4b4
VM/UNIX (Berkeley Version 2.1) 1/5/80
real mem  = xxx
avail mem = yyy
#
```

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.

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. 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

```
# chk /dev/rrp0a
```

which abbreviates

```
icheck /dev/rrp0a
dcheck /dev/rrp0a
```

The output from *chk* should look something like:

```
icheck /dev/rrp0a
/dev/rrp0a:
files    153 (r=111,d=12,b=8,c=22)
used   1065 (i=27,ii=0,iii=0,d=1038)
free   6558
missing        0
dcheck /dev/rrp0a
/dev/rrp0a:
    entries  link cnt
1       0       0
```

The diagnostic from *dcheck* is normal, as inode number 1 is reserved for placement of bad blocks, but currently unimplemented.

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 RP-06 use "43147" rather than "145673" in the procedure below.)

```
# /etc/mkfs /dev/rrp0g 145673  (create empty user filesystem)
isize = 65496                      (this is the number of available inodes)
m/n = 3 500                    (freelist interleave parameters)
# /etc/mount /dev/rp0g /usr    (mount the usr filesystem)
# cd /usr                      (make /usr the current directory)
# cp /dev/rmt5 /dev/null       (skip first tape file (tp format))
# cp /dev/rmt5 /dev/null       (skip second tape file (root))
# tar xbf 20 /dev/rmt1         (extract the usr filesystem)
# cd /                         (back to root)
# /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

```
# chk /dev/rrp0g
```

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

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

again.

Making a UNIX boot floppy

The next thing to do is to make your console floppy into a UNIX boot floppy, by placing some files on it using *arff*(1). If you don't have an extra console boot floppy, this is a good time to make one; see *flcopy*(1), and use it before playing with *arff.*

Once you have a duplicate copy of the console floppy in the machine, do the following

```
# cd /usr/src/sys/sys/floppy
# arff r *
```

This will put a copy of each file in the directory /usr/src/sys/sys/floppy onto the floppy. You should now be able to reboot using the procedures in *boot*(8). Try this after saying

```
# sync
```

which allows the system to initiate all i/o before you halt the CPU. It is traditional to say

```
# sync
# sync
```

to give the system a little time, and to then reboot.

Taking the system up and down

To bring the system up to a multi-user configuration after a boot 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.vm*(8) and *ttys*(5). 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 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.vm(8)* for more information.

Adding devices

The UNIX system running is configured to run with the given disk and tape, a console, and 8 DZ11 lines. This is probably not the correct configuration. You will have to correct the configuration table to reflect the true state of your machine.

Before you mess with the source for the system it is wise to make a backup. The following will do this:

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

This allows you to find out what you have done to the distribution system by later running the command *diffdir*(1), comparing these directories.

N.B.: Note that the system header files in /usr/src/sys/h are linked to 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 not break these links.

There are certain magic numbers and configuration parameters embedded in various device drivers that you may have 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. Also, if the devices's interrupt vector address(es) are not currently known to the system (this is likely), then the file /usr/src/sys/sys/univec.c must be modified appropriately: namely, the proper interrupt routine addresses must be placed in the table 'UNIvec'. Use the DZ11 as an example (as distributed, the DZ11 vectors are assumed to be at locations c0 and c4 (hexadecimal)).

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" 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.

The system comes with 4 drivers which we are using: A KL/DL driver *kl.c,* a Versatec printer/plotter driver *vp.c,* and two copies of a (old and simpleminded) UNIBUS disk driver, named *rm.c* and *rp.c.* These last two are not in the most aesthetically pleasing of shapes, but are fully functional. There is also an RK driver *rk.c,* but we are not using it, and it may need a little work.

The disk and tape drivers for the MASSBUS devices (hp.c, ht.c) are set up to run 1 drive and should be changed if you have more.

Now, make sure you add any new drivers which you have to the list of DRIVERS here, and to the FILES and CFILES variables so they will be compiled and included in listings. You can also delete drivers which you don't need from FILES and CFILES, or change the code so that nothing will be compiled by using a "#ifndef".

The *makefile* has several useful entry points:

clean	Cleans out the directory, removing **.o** files and the like.
lint	Runs lint on the system; the system was almost lint-free as sent to you. See *linterrs* for the remaining *lint* when it was distributed.
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	Produces a nice listing of most everything in the system directory in a canonical order.
symbols.sort	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.

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.† As distributed, the system is tuned for a fairly large machine. (There are also tunable constants in the file /usr/src/sys/h/vm.h but ignore them for the time being.)

To generate a new VM/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.

```
# make clean
# make depend
# make
```

and when this works

```
# make lint
```

to discover any residual bugs.

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 as /oldvmunix (and perhaps even a /oldervmunix) to guard against disaster.

Special Files

Next you must put in special files for the new devices in the directory /dev using *mknod*(1). Print the configuration file /usr/src/sys/sys/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 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 driver uses the 0100 bit of the minor device number to indicate whether or not to interleave a filesystem across more than one physical device. See *hp*(4) for more detail. The ht driver uses the 04 bit to indicate whether or not to rewind the tape when it is closed. The 010 bit indicates the density of the tape on TE16 drives. Again, see *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 '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 first 2 characters. In fact, we use the following convention: ''ttyN'', with N a number for normal DZ ports. ''ttydX'' with X a single character (starting from 0) for dialups, ''ttykX'' with X a single letter for KL ports, and ''console'' (abbrev ''co'') for the console. This works out well.

The files console, tty0-tty7, mem, kmem, kUmem, floppy and null are already correctly configured, as are special files for the default paging are /dev/drum, and the raw and block versions of the root and /usr file systems.

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.

Whenever special files are created, care should be taken to change the access modes (*chmod*(1)) 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*(1) to create any new filesystem 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 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. This should be large enough for most sites. You can change this if local wisdom indicates this is not good.

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. 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*(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 the root filesystem and system binaries (/usr), the temporary files and paging activity (/tmp and /dev/drum), and the user files among three disk arms. We will discuss such considerations more below.

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 (*dump*(1)) to magtape, use *mkfs*(1) to create the new filesystem, 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 filesystem 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 *filesy*(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*(1).
3. Bring the system down and mount the new pack.
4. Load the standalone versions of *mkfs*(1) and *restor*(1) 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
```

to correspond to your system, and then recompile and install *getty.vm*(8) via:

```
# cd /usr/src/cmd
# DESTDIR=/
# export DESTDIR
# Admin/mk getty.vm.c
```

This will arrange for an appropriate banner to be printed on terminals before users log in.

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 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 filesystems for consistency (*icheck, dcheck* (1)). It is quite important to execute *sync*(8) before rebooting or taking the machine down. This is done automatically every 30 seconds by the *update*(8) program when a multiple-user system is running, but you should do it anyway to make sure.

Dumping of the filesystem 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). See the scripts /etc/dumpusr and /etc/dumproot which we use to dump our file systems. You should arrange to do a towers-of-hanoi dump sequence; we tune ours so that each file is dumped twice and kept for at least a week in most every case. We take fully dumps every two to three weeks (and keep these indefinitely). Note that /etc/dumpusr maintain the file /etc/lastdumpdone. This can be printed out at login by people who can easily then start dumps if one is needed.

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.

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 is accomplished by using a directory in /usr/src/cmd, called Admin, which contains two files: mk and destinations. The file mk is a shell script for recompiling files in /usr/src/cmd. For instance, to recompile “date.c”, all one has to do is

```
# cd /usr/src/cmd
# Admin/mk date.c
```

this will place a stripped version of the binary of “date” in /usr/dist3/bin/date, since date normally resides in /bin, and Admin is building a file-system like tree rooted at /usr/dist3. You will have to make the directory dist3 for this to work. It is possible to use any directory for the destination, it isn’t necessary to use the default /usr/dist3. You can set the default 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
# Admin/mk * > Admin/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/Admin
# mk ALIASES
# mk 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. You should modify *vmstat*(1m) so that it prints out disk statistics for the disks you have, changing the headers to something appropriate for your system. Examine the definitions of DK_UNIT in the disk drivers supplied to see how the code in *vmstat* and *iostat* and the system correspond.

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, most jobs should be either running (RQ) or sleeping (SL), 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 (about 60%).

If the system is busy, then the number of active jobs may be large, and several of these jobs may often be in disk wait (DW). 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 in disk wait (DW) or page wait (PW), 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).

If the system is very heavily loaded, or if you have very little memory relative to your load (512K 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.

Tunable constants

There is a modicum of tuning available in the paging replacement algorithm 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/vm.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 remaining jobs in a strictly round-robin fashion, based on core residency time. 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 l2'') accumulated by the pageout demon will show how much overhead it is generating.

The system, as distributed, runs the replacement algorithm whenever less than 1/8 of the total user memory is free. This is done starting with a 30 second revolution time of the clock algorithm and increasing to a 20 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
/m 0 #ffffffff
lotsfree/D
---adb prints value of lotsfree---
/W 0t100
```

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

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.

In our system we currently have 1.75M bytes of memory and 2 disks: an RP06 and an AMPEX 9300. We run with the root, /tmp, and paging activity on the RP-06, while the /usr file system and user files are on the

9300. This gives a fairly even split of file activity in our environment.

A split such as this is a good initial guess if you have two arms. If you are fortunate to have three arms, you can try splitting the files up various ways. 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 area. Place the user files and the /usr directory as space needs dictate and experiment with the other, more easily moved file systems.

Finally, when you have your configuration worked out, you should set the constant MAXPGIO based on the maximum number of transfers you can expect from your paging device. The system assumes that if more transfers than this occur, then the system is overloaded, and unless there is a reasonable amount of free memory, it then begins to swap. The swap scheduler also consider jobs small if their size when they were swapped out was less than twice this constant. Such small jobs have a better chance of getting swapped back in when the core situation is tight, since they can be expected to be able to run in a small number of page frames.

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 4M bytes each, and the stack segment size is limited to 512K bytes. These can be increased by changing the constants MAXTSIZ, MAXDSIZ and MAXSSIZ in the file /usr/src/sys/h/vm.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.†

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* and *Syssize* in /usr/src/sys/sys/locore.s

The 4M byte limitation on individual segment size is enforced by the constants MAXTSIZ, MAXDSIZ and MAXSSIZ for the text, data and stack segments respectively. These can be increased, given the availability of adequate amounts of swap space, up to 16M bytes. In order to increase the size of the stack or data segments beyond 16M, you will have to increase the amount of space which can be mapped by the corresponding disk map. To increase, for instance, the maximum segment size to 32M bytes it would be adequate to double both the initial and maximum sizes for the diskmap. Thus defining (in /usr/src/sys/h/dmap.h)

```
#define    DMMIN      32
#define    DMMAX    8192
```

(i.e. a minimum segment size of 16K bytes and a maximum size of 4M bytes) would allow the 16 diskmap slots to map 32M bytes.

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 system buffers is limited to be less than 64 because of the way that the MASSBUS adaptor map registers are initialized. The construct ‘‘(128<<9)’’ appears in the /usr/src/sys/sys/mba.c and /usr/src/sys/sys/hp.c code, silently enforcing this restriction.

† Recovery from running out of paging area is currently not handled gracefully: the system panics.

Scaling down

If you have less than 1M byte of memory you may wish to scale the paging system down, by reducing some fixed table sizes not directly related to the paging system. For instance, we have raised NBUF from 32 to 48, NCLIST from 150 to 500, (also increasing the basic clist block size CBSIZE from 12 to 28) and NPROC, NINODE and NFILE for a fairly large system from the way they were distributed for UNIX/32V. We also increased TTLOWAT (from 50 to 125) and TTHIWAT (from 150 to 650.) If you pull NCLIST down, you should adjust these also. You can use *pstat*(1m) to find out how much of these structures are typically in use. Although the document is pretty much obsolete 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.

Odds and Ends

The programs dump, icheck, quot, dcheck, ncheck, and df (source in /usr/source/cmd) should be changed to reflect your default mounted filesystem devices. Print the first few lines of these programs and the changes will be obvious. You will probably want to amend some of /usr/lib/crontab, and will certainly want to add to /usr/lib/Mail.rc.

You should periodically examine the file /usr/adm/messages, which acts as a system error log (see dmesg(1m)). In particular, memory controller errors are checked for every 10 minutes and a diagnostic is produced (printing memory controller register C) if there were any errors.

Good Luck

William N. Joy
Ozalp Babaoglu