REGENERATING SYSTEM SOFTWARE

For UNIX/32V

Thomas B. London

John F. Reiser

Introduction

This document discusses how to assemble or compile various parts of the UNIX/32V® 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.

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).
libm	Source for the mathematical library.
libnm	Source for the assembler language 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/Admin
Mk cmd_name.c
```

suffices. (Cmd_name is the name of the command you are playing with.) The result of the Mk command will be an executable version, copied to /bin (or perhaps /etc or other places if appropriate). If you want the result placed somewhere else, the command

```
cd /usr/src/cmd/Admin
DESTDIR=mydir Mk cmd_name.c
```

where mydir is a full pathname of some destination directory (e.g. /usr/tbl/newcmds), will compile the command and place the result in mydir/bin (or perhaps mydir/etc or mydir/usr/bin, 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	The program is compiled and loaded; the executable is left in the current directory.
make install	The program is compiled and loaded, and the executable is installed.
make clean	Everything is cleanup; for example .o files are deleted.

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

Alternately, to compiler and install a subdirectory command, one may perform the following

```
cd /usr/src/cmd/Admin
Mk cmd_name
```

which combines all three of the above make options.

The Assembler

The assembler consists of one executable file: /bin/as. The source files for /bin/as are named '/usr/src/cmd/as/as?.c'. Considerable care should be exercised in replacing 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.

The C Compiler

The C compiler consists of six 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 C compiler proper is '/lib/ccom'; '/lib/c2' is the optional assembler-language 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 >temp0
/lib/ccom temp0 temp1
as temp3
ld /lib/crt0.o a.out −lc
```

The source for the compiler proper is in the directories /usr/src/cmd/mip and /usr/src/cmd/pcc. The /usr/src/cmd/mip directory contains files which are (relatively) machine independent; the machine dependent files reside in the directory /usr/src/cmd/pcc. The compiler is 'made' by the makefile (see make(1)) in the directory /usr/src/cmd/pcc. To make a new /lib/ccom use

```
cd /usr/src/cmd/pcc
make
```

which produces the compiler (named /usr/src/cmd/pcc/comp). Before installing the new compiler, it is prudent to save the old one someplace.

In a similar manner, the optimizer phase of the C compiler (/lib/c2) is made up from the files c20.c, c21.c, and c22.c together with c2.h. Its loss is not critical since it is completely optional.

UNIX

The source and object programs for UNIX are kept in two subdirectories of */usr/src/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 subdirectory *sys* is the rest of the system.

The file *conf.c* contains the tables which control device configuration of the system. *Locore.s* specifies the contents of the interrupt vectors, and all the machine-language code in the system.

To recreate the system, use

```
cd /usr/src/sys/sys
make unix
```

See 'Setting Up UNIX' for other information about configuration and such.

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

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

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.

To install a new device driver, copy its source to /usr/src/sys/sys, and edit the 'makefile' and the file 'loadall' if necessary (see make(1)).

Next, the I/O interrupt fielding mechanism must be modified to properly handle the new device. If the device is connected via the UNIBUS, then one only need add the device's interrupt handling routine address(s) in the proper position in the table 'UNIvec' in the file /usr/src/sys/sys/univec.c. 'UNIvec' should be modified by placing a pointer to a callout routine in the proper vector. Use some other device (like the DZ11) as a guide. Notice that the entries in 'UNIvec' must be in order. Bits 27-31 of the vector address 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 dz11's).

If the device is connected via the MASSBUS, then /usr/src/sys/sys/univec.c is not to be modifed. Instead, code must be added to /usr/src/sys/sys/locore.s to actually transfer to the interrupt routine. Use the RP06 interrupt routine 'Xmba0int' in locore.s as a guide. 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 new device is to add it to the configuration table /usr/src/sys/sys/conf.c. This file contains two subtables, 'bdevsw' and 'cdevsw', one for block-type devices, and one for character-type devices. Block devices include disks, 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
make libc.a
make install
make clean
```

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
for i in csu/*.s
do
        j=‘basename $i .s‘
        as -o $j.o $i
        mv $j.o /lib
done
```

or, if you need only redo one routine,

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

where x is the routine you want.

Other Libraries

Likewise, the directories containing the source for the other libraries have makefiles.

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 /usr/sys/h/param.h as manifests ('#define's). 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.

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 specifies the number of page-frames of real memory at this installation. It is currently set at 1024 (512K bytes), and should be increased if you have more (otherwise the additional memory will not be utilized).

MAXUMEM
This sets an administrative limit on the amount of memory a process may have. It is currently set at MAXMEM-128 (i.e. 896). It will be increased automatically by increasing MAXMEM. Note, however, that the current upper bound on MAXUMEM is 128*12 (i.e. 1536) which limits user process space to 768K bytes.

PHYSPAGES
This indicates the number of pages which can be represented on the memory freelist. Its current value is 2048, and is sufficient for systems of up to one megabyte. If this value is too small (i.e. more memory than freelist) then system will only use PHYSPAGES page frames.

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.

SMAPSIZ The number of fragments that secondary (swap) memory can be broken into. This should be big enough that it never runs out. The theoretical maximum is twice the number of processes, but this is a vast overestimate in practice. 70 seems enough.

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 12 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 desired frequency of the system clock (e.g., 50 for a 50 Hz. clock). The current value is 60 (i.e. 60 Hz. clock).