UNOS 1017

Bell Laboratories

subject: Setting Up UNIX - Issue Three
Regenerating System Software
Issue Three

date: September 19,1975

from: I. A. Winheim
MF-75-8234-194

MEMORANDUM FOR FILE

The attached documents describe the procedure used to install standard UNIX systems on a PDP 11/40 or 11/45, and the steps necessary to regenerate sections of the system software. These procedures apply to the UNIX Support Group's (USG) UNIX Release 2 and the UNIX Operating System Program Generic, PG-1C30E, Issue 1.

The attached documents are revised versions of "Setting Up UNIX" and "Regenerating System Software" by K. Thompson and D. M. Ritchie. Two types of modifications were made to the original documents. The first makes the documents apply to systems generated by the USG SYSGEN procedure which eliminates certain steps previously executed by hand. For example the SYSGEN shell executes the "tlrc" shell file on all 11/40 machines and on 11/45 machines without the floating point option. For more details on SYSGEN see "Creating a New File System" by I. A. Winheim. The second type of modification incorporates some of the knowledge gained by the USG during the trouble-shooting of new systems. Section 9, Multiple Users, of "Setting Up UNIX" is one example of this type of modification.

The information in the attached documents is meant to apply only to systems generated by the USG, although some of the information is universal to UNIX.

I. A. Winheim

MH-8234-IAW-nroff

SETTING UP UNIX - Issue Three

I. MANIFEST

Enclosed are:

1. The UNIX software on magtape or disk pack.
2. "Regenerating System Software"
3. Your configuration.
4. Sysgen shell procedure used.
5. Proto file
6. conf.c
7. low.s
8. name list of your system

II. SYSTEM ON MAGTAPE

If your system is on magtape, perform the following bootstrap procedure to obtain a disk pack.

1. Mount magtape on drive 0 at load point.
2. Mount formatted disk pack on drive 0.
3. Key in and execute at 100000
   for a TU10 magtape drive.

   012700
   172526
   010040
   012740
   060003
   000777

   The tape should move and the CPU loop.
   (This is not the DEC bulk ROM for tape.)

OR

If your machine is an 11/45 or an 11/70 this can be shortened. (On the 11/70 there are more console switches and the addresses below are for the 11/45 machines. To convert the addresses to 11/70 simply set the unspecified high order switches to the up position.)

For TJU16
Make sure the following addresses are zeroed

772442
772444
772446

Halt and load address 772472
Set switches to 001300 and deposit
Load address 772440

Set switches to 000071
Enable and deposit (in that order)

For TU10
Halt and load address 772522
Set switches to 060003
Enable and deposit (in that order)

4. Halt and restart the CPU at 0.
(Load address 000000 -
set switches to 773030 to come up single user
Enable and start)
The tape should rewind.
The console should type '='.
5. Copy the magtape to disk by the following:

```
=                                                    (machine)
list<                                                (you) (<=CR/LF)
(This will list all the files on the
 front of the tape. The last line
 will look like:
    tape size = NN (decimal)
 where NN is a number in decimal.)
=                                                    (mch)
copy<                                                (you)
p for RP; f for RF; k for RK; 4 for RP04             (mch)
p                (you) (or k,f,or 4 as the case may be)
m for TM11; u for TU16; c for TC11                   (mch)
m                (you) (or u or c as the case may be)
disk offset<                                         (mch)
0<                                                   (you)
tape offset<                                         (mch)
NN<              (you) (where NN is gotten from list)
        (The tape should move)
count<                                               (mch)
4000<            (you) (the tape moves more)
=                                                    (mch)
```

6. At this point you have a UNIX system on disk.
It is advisable to come up on the disk from
the tape at this point. This is done in order
to copy the correct boot program onto the front
of the pack being used as the root. (Due to the
size of the new boot programs a utility package
containing the necessary boots would not fit
on block zero.) Make sure the switches are set
to 773030 and respond to the "=" with:

```
hpboot<      OR      rkboot<          OR      rpboot<
4                    k                        p
             (note CR echoed by mch)
/unix<               /unix<                   /unix<
```

There will be no prompts after the boot program

is typed. The only way to tell that you are on the right track is if the machine echoes a carriage return. If no CR is echoed the wrong character was typed and this boot will not work. If this is the case repeat the above booting procedure from the tape (steps 3, 4, and 6). UNIX is now running on the disk pack. You know this by the console message "UNIX Release 3". A special file must be made in directory /dev for the root device. Section VI (Special Files) describes this procedure. NOTE all the special files may be entered at this time or they may be entered at a later time. With this completed, copy the correct boot (resides in the directory /usr/mdec/tu) to the device with the cp-I command. For example use one of the following:

```
cp /usr/mdec/tu/hpboot /dev/hp0
cp /usr/mdec/tu/rkboot /dev/rk0
cp /usr/mdec/tu/rpboot /dev/rp0
```

for theRP04, RK04/05, and the RP03 respectively. The correct boot program is now on block zero of the root device and any future boots can come up directly on this device.

III. SYSTEM ON DISK

You now have a UNIX system on disk and have moved the proper boot program to the front of the root pack (see above). You may want to extract and execute UNIX from this pack, if so key in and execute the following program at location 100000.

RP03	RK	RP04
012700	012700	
176726	177414	(see
005040	005040	ROM
005040	005040	below)
005040	010040	
010040	012740	
012740	000005	
000005	105710	
105710	002376	
002376	005007	
005007		

NOTE: These programs correspond to the DEC bulk ROM for RP and RK respectively.

OR

On an 11/45 or an 11/70 these programs can be shortened. Note that the 11/70 addresses must be converted as they were with the tape.

For an RK
- Halt and load address 777404
- Set switches to 000005
- Enable and deposit (in that order)
- Halt and load address 000000
- Set switches to 773030 (single user)
- Enable and start

For an RP03
- Halt and load address 776716
- Deposit a non-zero number
- Halt and load address 776714
- Set switches to 000205
- Enable and deposit (in that order)
- Halt and load address 000000
- Set switches to 773030 (single user)
- Enable and start

For an RP04
- Halt and load address 765000
- Set switches to 000070
- Enable and start

For an RS04
- Halt and load address 765000
- Set switches to 000100
- Enable and start

The CPU should loop. If the device being booted is the RP04 or the RS04 and you wish to come up single user set the console switches to 773030 at this point. At this point type p for RP or k for RK or 4 for RP04 or s for RS04. A CR will be echoed. Now type /unix followed by a CR. After a delay, UNIX will come up by announcing

(SINGLE USER)	(MULTI USER)
UNIX Release x	UNIX Release x
mem = xxx	mem = xxx
	:login:

The mem message gives user available memory in .1K units. The number should be 120 (for 12K) or larger for UNIX to fully support all of its software.

UNIX is now running. The "UNIX Programmer's Manual" now applies. (Below references to section X entitled Y of the programmer's manual are given (Y-X).) The only valid user names are root and bin. The root is the super-user and bin is the owner of nearly every file in the file system.

Before UNIX is turned up completely, a few configuration dependent exercises must be performed. First login (login-I) as root. (This is not necessary if the system was brought up in single user mode.)

IV. FLOATING POINT

If your machine does not have the 11/45 floating point option, (this includes all 11/40 systems) the following command was executed before your system was delivered.

```
sh /usr/sys/conf/flrc
```

This sequence recompiled the C compiler and the debugger not to use floating point. It also installed the floating point interpreter in the standard library and reloaded basic, and fortran to have their floating point interpreted.

V. TIME CONVERSION

If your machine is not in the eastern time zone, you must edit (ed-I) the subroutine /usr/source/s4/ctime.c to reflect your local time. The variable timezone should be changed to reflect the time difference between local time and GMT. For EST, this is 5*60*60; for PST it would be 8*60*60. For example to convert to pacific time:

```
ed /usr/source/s4/ctime.c
/5\*/s//8*/
```

Three lines farther, is the name of the standard time zone. It should be changed to reflect local standard time. Thus continuing the above example

```
.3s/\e/\p/
```

The next line is the daylight time zone name, if any.

```
.1s//\p/
w
q
```

NOTE that these two names are in upper case and will have to be edited using escapes (dc-IV) as shown above. Two lines farther is the daylight flag. It has the value 1 which causes the time to shift to daylight savings automatically between the last Sundays in April and October. Normally this will not have to be reset. After ctime.c has been edited (make sure it is rewritten), execute the following command.

```
sh /usr/sys/conf/tmrc
```

This will recompile all the programs that use the local time conversion. This execution takes about 7 minutes, and requires that the floating point corrections have been installed if needed. If the editing of ctime.c was botched, there will be diagnostics from its compilation.

Set the current date (date-I).

VI. SPECIAL FILES

At this point, it would be wise to read all of the manuals and to augment this reading with hand to hand combat. It might be instructive to examine the shell command files for floating point and time conversions. The rest of the conversion assumes that sections I, IV, V, VII, and VIII of the programmer's manual have been read.

Next you must put all of the special files in the directory /dev using mknod - VIII. Print the configuration file:

```
cat /usr/sys/conf/conf.c
```

This is the major device switch of each device class (block and character). The block and character device configuration tables have been standardized by the UNIX Support Group. An entry for all common devices in both device classes has been made. If a device was not present in the system the entry in the bdevsw (block device switch) or cdevsw (character device switch) table would read:

```
&nodev,    &nodev,    &nodev,    &nodev,    &nodev,
```

The MAJOR device number for this entry would therefore not be associated with any specific file. The MAJOR device number is selected by counting the line number (from zero) of the device's entry in the device switch table, or, since conf.c has been standardized, by taking the number from the following table.

Block Devices

conf.c	TYPICAL NAME	MAJOR	device
rk	rk0	0	RK moving head disk
rp	rp0	1	RP moving head disk
rf	rf0	2	RS/RF fixed head disk
tm	mt0	3	TM/TU magtape
tc	tap0	4	TC/TU DECtape

Character Devices

conf.c	NAME	MAJOR	device
kl	tty8	0	console tty
pc	ppt	1	paper tape
lp	lp	2	high speed line printer
dc	ttyx	3	async serial line interface
dh	ttyx	4	async serial line multiplexer
dp		5	synchronous interface
dj	ttyz	6	async serial line multiplexer
dn		7	alternate console
mem	mem	8	memory
rk	rrk0	9	RK moving head disk
rf	rrf0	10	RS/RF fixed head disk for
rp	rrp0	11	RP moving head disk physical
tm	rmt0	12	TM/TU magtape I/O

Three devices have been added to the configuration table. These are RP04(np), RS04(hs), and TU16(ht) where the parenthesized names are the ones that appear in CONF.C. If present in the configuration they would start at Block major device number 5 (five) and Character major device number 13 (thirteen). The major device numbers can not be gotten from the above table so they must selected by counting the line number (from zero) in the device itch table.

The block special devices are entered into directory /dev first by executing the command:

/etc/mknod /dev/NAME b MAJOR MINOR

where NAME and MAJOR can be gotten from the preceeding table. The only exception being the TU16. If this device is being entered into /dev it should be called 'mt' not 'ht' as it is in the configuration tables. This is because the command TP(I) uses the pathname '/dev/mt' to execute. The MINOR device is the drive number, unit number, or partition number as described under each device in section IV of the programmer's manual. The last digit of the name (all given as 0 in the table above) should reflect the minor device number. For tapes where the unit is dial selectable, a special file may be made for each possible selec-

tion. If an RK, RP, TM, and TC were configured in the system the following special files might be created..

```
/etc/mknod /dev/rk0 b 0 0
/etc/mknod /dev/rk1 b 0 1
/etc/mknod /dev/rp0 b 1 0
/etc/mknod /dev/rp1 b 1 1
/etc/mknod /dev/mt0 b 3 0
    . . .
/etc/mknod /dev/mt7 b 3 7
/etc/mknod /dev/tap0 b 4 0
    . . .
/etc/mknod /dev/tap7 b 4 7
```

This sets up file rp0 and rp1 each as a half of one RP03 drive. Files rk0 and rk1 correspond to two RK03/RK05 drives. The tape files will handle up to eight handlers dialed to any unit.

The same goes for the character devices. Here the names are arbitrary, except that devices meant to be used for teletype access should be named /dev/ttyX, where X is any character. The console is already in as:

```
/etc/mknod /dev/tty8 c 0 0
```

All 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. The open and close routines are the same as the standard ones for the device but there are special read and write routines. For example, to install raw magtape, the line

```
&tmopen, &tmclose, &tmread, &tmwrite, &nodev,
```

was added to the character device switch to make the appropriate special file for it (our name is "/dev/rmt0").

Another useful special file is "/dev/mem," which is configured in the system as distributed. The appropriate line to the character device switch is

```
&nulldev, &nulldev, &mmread, &mmwrite, &nodev,
```

and the name of the special file should be "/dev/mem." This file must exist for the ps command (I) to work.

VII. MOUNTED FILE SYSTEMS

If there are to be more file systems mounted than just the root, use mkfs-VIII to create the new file system and put its mounting in the file /etc/rc (see init-VII and mount-VIII). For example, to set up a second RK as a mounted file system.

```
/etc/mkfs /dev/rk1 4872
/etc/mount /dev/rk1 /mnt
ed /etc/rc
$a
/etc/mount /dev/rk1 /mnt
.
w
q
```

VIII. NEW USERS

Install new users by editing the password file /etc/passwd (passwd-V). You'll have to make current directories for the new users in either /usr or /mnt and change their owners to the newly installed name. Login as each user to make sure the password file is correctly edited. For example:

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

This will make a new login entry for joe. His default current directory is /mnt/joe which has been created.

IX. MULTIPLE USERS

If UNIX is to support simultaneous access from more than just the console teletype, the file /etc/ttys has to be edited. Each line in this file describes a typewriter channel potentially available for logging-in. Each line has 3 characters: the first is a 0-or-1 flag which, if 1, causes the typewriter to have a login process for it and if 0 to be ignored. The second character is the last character of the typewriter name (E.g. "x" for "/dev/ttyx".) The third character is a digit which selects which of several programs is to be executed when a data connection is established on the line. The list is built into the init program

(VII) and currently the only program is /etc/getty and the only digit is 0. The /etc/ttys file as distributed contains lines for typewriters 0-9 and a-d but all except tty? are turned off. For some historical reason tty0 is the name of the console typewriter. To add new typewriters 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). Note that init.c will have to be recompiled if there are to be more than 20 typewriters. Also note that if the special file is inaccessible when init tries to create a process for it, the system will thrash madly trying and retrying to open it.

Another problem occurs on dc interfaces that are wired to run at any of four speeds. Within /usr/sys/dmr/dc.c there is a table (dcrstab) containing speeds and two bits for hardware. There are four speeds flaged as 0,1,2,or 3 which should correspond to the hardware. The dc driver recieves a speed for a tty from a program called getty. The speed is a displacement into the dcrstab table. If the displacement falls on a zero slot an error condition is noted, otherwise the speed of the entry is used.

Getty cycles through speeds and login messages as specified in the itab table. The user depresses the interupt or the break key to tell getty to try the next speed.

The itab entry looks as follows:

```
itab:
        itn300;tnmes;tn300
        itty37;ttymes;tty37
        0
```

This represents the initial speed and parity settings, the address of the login message, and the subsequent speed and parity settings respectively for 300 baud and tty model 37 terminals.

A tty entry is as follows:

```
itty37:
        .byte   5,5             /speed 5
        0,340                   /any parity, raw
```

The first two bytes represent the displacement into the dcrstab and dctstab tables in the dc driver. A 5, for example, is 150 baud speed 1. The second entry (0,340) is the bit setting for the parameters such as parity, raw mode, echo, as described in STTY(II) of the Reference Manual.

The entries in the dc driver are for selectable speeds 134.5, 150, 300, 1200 baud and correspond to a DC11-AG. Any other DC11 requires that the entries in the dcrstab and dctstab tables be changed.

Another common modification is to build a new getty for use with a "funny" terminal (for example a CRT). This is done by adding entries to the itab table as described above. For example to add 1200 baud entry the following procedure could be used:

```
chdir /usr/source/s1
cp getty.s ngetty.s
ed ngetty.s
/itab/
a
it1200; tnmes; tn1200
.
/ttymes/
i
it1200:
        .byte 9,9
        0;340           /any parity, raw
tn1200:
        .byte 9,9
        0;10310         /any parity, echo, no tab, del
.
w
q
as ngetty.s
ld a.out
mv a.out /etc/ngetty
```

There are now two getty files and init.c must be edited to show this. Within /usr/source/s1/init there is a list of getty file names and the number of getty files. To do this change ncom, and the table com:

```
define ncom equal to the number of getty files
    *com[ncom]
        add the new getty file name
Then compile init.c as
    cc -n -s init.c
    mv a.out /etc/init
```

The final file to edit is /etc/ttys to make sure all the ttys are enabled. Remember the third digit of each entry in the ttys file is the displacement into the com table of init.

It is possible to add or drop typewriters without rebooting the system by sending a "hangup" signal to the initialization process; init catches this signal and rereads the /etc/ttys file. It then terminates processes on those lines that are to be dropped and creates them for those that are to be added. It is careful not to disturb processes on lines that are not being changed. When this signal is sent init examines the console switches and obeys them as if the system was being rebooted; that is, if the switches indicate a single-user system, everyone is logged out, and conversely other switch settings cause a

multiple-user system to be continued or started. The hangup signal is sent by executing the command

```
kill -1 1
```

the super-user. The "-1" indicates the hangup signal; the "1" is the process number of init. [Note: this whole feature is new and is not documented elsewhere. Init used to have the names of the typewriters built into it and had to be recompiled for each change in typewriter configuration.]

FILE SYSTEM HEALTH

Periodically (say every day or so) and always after a crash, you should check all the file systems for consistency (check-VIII). It is quite important to execute sync (VIII) before rebooting or taking the machine down. This is done automatically every 30 seconds by the update program (VIII) when a multiple-user system is running, but you should do it anyway to make sure.

Dumping of the file system should be done regularly, since once the system is going it is very easy to become complacent. It should be pointed out that our RP controller has failed three times, each time in such a way that all information on the disk was wiped out without any error status from the controller. Complete and incremental dumps are easily done with the dump command (VIII) but restoration of individual files is painful. Dumping of files by name is best done by tp (I) but the number of files is limited. Finally if there are enough drives entire disks can be copied using cp-I, or preferably with a program provided by you which uses raw I/O to copy a track or so at time.

XI. ODDS AND ENDS

The programs dump, check and df (source check.c, dump.c, and df.c all in /usr/source/s1) have to be changed to reflect your default mounted file system devices. Print the first few lines of these programs and the changes will be obvious.

The source that is provided on the file system is quite space consuming. You will want to probably dump it offline (with dump-VIII, tp-I or cp-I as above) and remove it. Care should be exercised in deleting any files in the following directories: /bin, /etc, /lib, /usr/bin, /usr/lib, /usr/games. Also the following files should not be deleted: /usr/fort/fc1, /usr/sys/unix, /dev/tty8.

The source of the system proper is kept in /usr/sys. Most of the subsystem source is kept in /usr/source. For a complete list of the file system, use check-VIII or du-I (with -1 flag).

If there are any programs that are not part of your system and you wish to retain a tape of them, this can be arranged .

REGENERATING SYSTEM SOFTWARE - Issue Three

I. INTRODUCTION

This document discusses how to assemble or compile various parts of the UNIX system software. This may be necessary if 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. It should be noted that in the system as distributed, there are quite a few commands that depend to some degree on the current configuration of the system; thus in any 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 ("disk free") command has built into it the names of the standardly present disk storage drives (e.g. "/dev/rf0", "/dev/rp0"). Df 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.

The greater part of the source files for commands resides in several subdirectories of the directory /usr/source. These subdirectories, and a general description of their contents, are

s1 Source files for most commands with names beginning with "a" through "l".

s2 Source files for most commands with names beginning with "m" through "z".

s3 Source files for subroutines contained in the standard system library, "/lib/liba.a" (see below).

Source files for the C library, "/lib/libc.a" (see below)

To regenerate most commands in the s1 and s2 directories is straightforward. The appropriate directory will contain one or more source files for the command. These will all have the suffix ".s" if the command is written in assembler language, or ".c" if it is written in C. The first part of the name begins with the name of the command. If there are several source files, the command name will be followed by a character which distinguishes the several files. it is typically "1", "2", ...; Sometimes the last is "x". For example, The "bas" command has source files (in s1) called "bas0.s", "bas1.s", ..., "bas4.s", "basx.s". In all cases, the lexicographical order of the distinguishing character is the order in which the source files should be compiled or assembled. Thus, for example, the way to reassemble a new "bas" is to say (in s1)

```
as bas?.s
```

Some of the assembly-language commands are completely stand-alone and require no application of the link editor ld (also loosely called the loader). Unfortunately there is no a priori way of determining which need to be loaded. A simple a posteriori method is to assemble the command as discussed above, then say

```
nm -u a.out
```

which will list the undefined external symbols. If any appear, the loader should be called by saying

```
ld a.out -l
```

A minority of the commands written in assembly language are coded so that their text portions are pure and they can be shared. (The most important of these is "ed"; others are "dc" and "write.") Such commands may (and in the case of the editor, should) be loaded (whether or not they need it for picking up library routines) by

```
ld -n a.out -l
```

One important command which needs slightly special treatment is "tp" which has to be loaded with the C library:

```
as tp?.s
ld a.out -l -lc
```

because it calls the C-language ctime subroutine.

It is not particularly easy to find out if an assembly-language command has a pure text segment. The simplest way is probably to look at the source and see if there are ".data" assembler operators surrounding system calls with pluggable arguments. It is probably not a bad idea to ignore the whole question except in case of the editor, where there are likely to be real gains text-segment sharing.

As it happens, there are no commands written in C (except those described below) which consist of more than one file. The command "com.c" can therefore be recompiled simply by saying

```
cc -n com.c
```

as above the "-n" indicates the desire to produce a object file which has a pure, sharable text segment. Since C produces pure code and the C library is pure, one might as well share.

Some of the most important commands are considerably more complicated to regenerate, and these are discussed specifically below. The contents of libraries are also discussed.

II. AS

The assembler consists of two executable files: /bin/as and /etc/as2. The first is the 0-th pass: it reads the source program, converts it to an intermediate form in a temporary file "/tmp/atm0?", and estimates the final locations of symbols. It also makes two or three other temporary files which contain the ordinary symbol table, a table of temporary symbols (like n:) and possibly an overflow intermediate file. The program /etc/as2 acts as an ordinary two-pass assembler with input taken from the files produced by /bin/as.

The source files for /bin/as are named "/usr/source/s1/as1?.s" (there are 9 of them); /etc/as2 is produced from the source files "/usr/source/s1/as2?.s"; they likewise are 9 in number. Considerable care should be exercised in replacing either component of the assembler. Remember that if the assembler is lost, the only recourse is to replace it from some backup storage; a broken assembler cannot assemble itself.

III. C

The C compiler consists of three files: "bin/cc", which expands compiler control lines and which calls the phases of the compiler proper, the assembler, and the loader; "/lib/c0", which is the first phase of the compiler; and "/lib/c1", which is the second phase of the compiler. The loss of the C compiler is as serious as that of the assembler.

The source for /bin/cc resides in "/usr/source/s1/cc.c". Its loss alone is not fatal. Provided that prog.c does not contain any compiler control lines, prog.c can be compiled by

```
/lib/c0 prog.c temp0 temp1
/lib/c1 temp0 temp1 temp2
as - temp2
ld -n /lib/crt0.o a.out -lc -l
```

If /bin/cc is lost, it can be recovered in this way, since it contains no compiler control lines.

The source for the compiler proper is in the directory /usr/c. The first phase (c0) is generated from the files c00.c, ..., c04.c, which must be compiled by C; c0t.s, which must be assembled; and c0h.c, which is a header file which should not be compiled but is a file included by the C programs of the first phase. The c0t.s program contains a parameter "fpp" which determines whether C is to used on a machine which has PDP 11/45 floating-point hardware; it should be set to 1 if so, 0 if not. In the standard system fpp is 1. To make a new /lib/c0, assemble c0t.s, name the output c0t.o, and

```
cc -n c0t.o c0[0-4].c
```

Before installing the new c0, it is prudent to save the old one someplace.

The second phase of C (/lib/c1) is generated from the C source files c10.c, ..., c12.c, the assembly-language program c1t.s, the include-file c1h.c, and a library of object-code tables called tab.a. To generate a new second phase, assemble c1t.s, call it c1t.o, and

```
cc -n c1t.o c1[0-2].c tab.a
```

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

The library of tables mentioned above is generated from the files regtab.s, sptab.s, cctab.s, and efftab.s. The order is not important. These ".s" files are not in fact assembler source; they must be converted by use of the cvopt program, whose source and object are located in the C directory. For example:

```
cvopt regtab.s temp
as temp
mv a.out regtab.o
ar r tab.a regtab.o
```

IV. FORTRAN

Probably because it is a very large subsystem written entirely in assembly language, Fortran is quite complicated to regenerate. On the other hand, Fortran is vital only to its own users; since none of the compiler nor any important part of the run-time system is written in Fortran, both can be regenerated in case of loss.

The fc command itself is essentially equivalent to a long shell command file; for a single source program prog.f, it amounts to saying

```
/usr/fort/fc1 prog.f
as - f.tmp1
ld /lib/fr0.o a.out /lib/filib.a -lf -lI
```

Thus, /usr/fort/fc1 is the compiler proper; fc1 leaves its output in the current directory in the file "f.tmp1". /lib/fr0.o is the runtime startoff. The file filib.a is the library of operators; Fortran is essentially interpretive, and operations such as "add floating variable to floating variable" are short routines loaded from the filib.a library.

/lib/libf.a (specified by the "-lf") is an archive file containing the language builtin functions plus a few others. The standard assembly language library (the "-l", or /lib/liba.a) is referenced by certain of the builtin functions (for routines like).

The source and object of the compiler are stored in subdirectories of the /usr/fort directory, named f1, f2, f3, f4, and fx. The first four represent putatively separable phases; the last contains subroutines used by several of the phases. Each directory contains an archive file with the object programs corresponding to the source programs in that directory; it is called f?o.a where "?" is the last letter in the directory name. reload Fortran from theses libraries, see the Shell command file /usr/fort/ld, which should contain

```
ld -u pass1 -u pass2 -u pass3 -u pass4 \
   f1/f1o.a f2/f2o.a f3/f3o.a f4/f4o.a fx/fxo.a -l
```

Each subdirectory should contain a Shell command file called "as" which assembles a particular file in that subdirectory; the one for f1 contains, for example,

```
as ../fx/fhd.s f1$1.s
mv a.out $1.o
ar r f1o.a f1$1.o
rm f1$1.o
```

so that the command

```
sh as 5
```

would assemble f15.s (preceded by the definition file /usr/fort/fx/fhd.s) and place it in the library for that subdirectory.

Actually we hope that no one will be required to make a new Fortran from the pieces, or fix it themselves. For those who are rious, we will say that phase 1 analyzes declarations, phase 2 does storage allocation, phase 3 code generation, and phase 4 puts out constants, code from format and data statements, and the actual storage-reserving code from variables.

V. UNIX

e source and object programs for UNIX are kept in /usr/sys and three subdirectories therein. The main directory contains several files with names ending in ".h"; these are header files which are picked up (via "#include ...") as required by each system module. The files lib1 and lib2 are libraries (archives) of (almost) all the object programs in the system. The file lib1 is made from the source programs in the subdirectory ken; lib2 is

made from the programs in subdirectory dmr. The latter consists mostly of the device drivers together with a few other things, the former is the rest of the system.

Subdirectory conf contains the programs which control device configuration of the system. Low.s specifies the contents of the interrupt vectors; conf.c contains the tables which relate device numbers to handler routines. A third program, mch.s, contains all the machine-language code in the system. It contains conditional-assembly flags which control whether the system will run on an 11/40 or /45, and whether the system supports the 11/45 floating-point unit. The file mch.s is quite long, but almost all of it is concerned with recovering after a stack violation in the 11/40.

To recreate the system, sysgen compiles conf.c and moves the output to /usr/sys/conf.o., assembles low.s and mch.s, and moves the output to /usr/sys/low.o and mch.o respectively. Sysgen then checks to see if floating-point is part of the configuration. If there is no floating-point FLRC is executed to set the floating-point parameters in /bin/db, /usr/source/s3/fp?.s, /lib/lib.a, /usr/fort, /usr/source/s1/bas?.s, and /bin/bas to 0 indicating no floating-point. Sysgen then changes directory to /usr/sys, and loads the whole system.

When the Sysgen is done,the new system resides on /dev/rp0 It can also be tested by putting it on tape (tp-I) and using tboot or mboot, or directly using uboot (boot procedures-VIII). When you have satisfied yourself that it works, it should be renamed /unix so that programs like ps (I) can use it to pick up addresses in the system.

To install a new device driver, compile it and place the object in lib2 if necessary. (All the device drivers distributed with the system are already there.) The device's interrupt vector must be entered in low.s. This involves placing a pointer to a callout routine and the device's priority level in the vector. As an example, consider installing the interrupt vector for DC11 number 2. Its receiver interrupts at location 320 and the transmitter at 324, both at priority level 5. Then low.s has:

```
. = 320^.
dcin; br5+2
dcou; br5+2
```

First, notice that the entries in low.s must be in order, since the assembler does not permit moving the location counter "." backwards. The assembler also does not permit assignation of an absolute number to ".", which is the reason for the ". = 320^." subterfuge; consult the Assembler Manual for the meaning of the notation. If a constant smaller than 16(10) is added to the priority level, this number will be available as the first argument of the interrupt routine. This stratagem is used when

several similar devices share the same interrupt routine.

At the end of low.s, add

```
        .globl  _dcrint
dcin:
        jsr     r0,call; _dcrint

        .globl  _dcxint
dcou:
        jsr     r0,call; _dcxint
```

The call routine saves registers as required and makes a C-style call on the actual interrupt routine (here _dcrint and _dcxint) named after the jsr instruction. When the routine returns, call restores the registers and performs an rti instruction.

To install a new device thus requires knowing the name of its interrupt routines. These routines are in general easily found in the driver; they typically end in the letters "int" or "intr." Notice 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/sys/conf/conf.c. This file contains two subtables, one for block-type devices, and one for character-type devices. Block devices include disks, DECtape, and magtape. All other devices are character devices. A line in each of these tables gives all the information the system needs to know about the device handler; the ordinal position of the line in the table implies its major device number, starting at 0.

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

For character devices, each line in the table specifies a routine for open, close, read, and write, and one which sets and returns device-specific status (used, for example, for stty and gtty on typewriters). If there is no open or close routine, "nulldev" may be given; if there is not read, write, or status routine, "nodev" may be given. This return sets an error flag and returns.

The above discussion is admittedly rather cryptic in the absence of a general description of system I/O interfaces. (see " The UNIX I/O System " by Dennis Ritchie) If you get to this point, we

suggest you examine the undocumented program /usr/sys/conf/mkconf.c. (this program will be documented in the near future)

The final step which must be taken to install a device is to make a special file for it. This is done by mknod (VIII), 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).