# Generating MERT Software

These instructions are intended for installations which have a MERT system running (see section 19) and want to (re-)generate MERT for one of the following reasons:

A. You have booted the MERT distribution system, and this is the first attempt at generating a customized MERT operating system.
B. An existing MERT installation wants to upgrade to the new Release 0.
C. An existing MERT Release 0 installation wants to make a change to the system, for example add a new driver or install a modification.

For ease of reference, paragraphs of this section are listed below in alphabetical order with the paragraph number preceding each directory or file.

| Paragraph | Name | Contents |
|---|---|---|
| 9 | adm | administrative programs. |
| 10 | appl | application tests for new MERT-UNIX features. |
| 13 | bdev | block and record device drivers. |
| 12 | cdev | character device drivers and system library. |
| 3 | conf.d | A file defining device channels. |
| 5 | elib | source files for */lib/libe.a*, the library routines to interface EMT's from supervisor processes to the kernel. |
| 15 | fmgr | MERT file manager files. |
| 20 | header.s | A file in *sgen* with global system options |
| 21 | kern | kernel, including memory manager and scheduler, and initialization process. |
| 4 | kfs | UNIX user programs which differ from those of standard of UNIX and some MERT file system utility programs (See *ukfs* for their UNIX file system counterparts). |
| 6 | klib | source files for */lib/libk.a*, the library routines to interface EMT's from kernel processes to the kernel. |
| 23 | mon | system monitor process and analysis programs. |
| 14 | pmge | process manager, nub process and termination supervisor (pkill). |
| 24 | publib | test programs for public libraries in user programs. |
| 7 | rlib | source for */lib/librr.a*, the library routines which interface the MERT-UNIX system calls to user programs. |
| 22 | sgen | files used to *sysgen* a new system. |
| 8 | slib | source for */lib/libs.a*, some general routines used by supervisor processes. |
| 11 | syslib | System Library |
| 16 | ukfs | UNIX file manager utilities (see *kfs*) |
| 17 | unix | MERT-UNIX supervisor. |
| 16 | unixfmgr | unix file manager. |

All of the above are directories or files in */src/mertsrc*. Also in */src/mertsrc* are a number of descriptor or header files (*.d files) which are included when compiling various MERT routines and programs.

**1. Preparing for Recompilation**

First, **back up your current system**, unless you have just come up on the distributed root system for the first time. Before you start doing anything to your current system (which has been in use for some time and therefore is assumed to contain some valuable information), you should back up your system. Preferably, make a disk dump (*dump-VIII*) of your entire disk (the one containing the root file system if you have more than one). Then start with a new disk pack onto which you restore (*restor-VIII*) the dump taken before. (Of course, you could do simply a disk-to-disk copy with *pcp-e*.)

You should be aware of a potential problem area regarding the disk layout as understood by the disk driver software. Previous MERT systems map the RP disks in ways different from Release 0. Release 0 follows the UNIX Generic 3 standard disk layouts as defined in the appropriate pages of Section IV, UNIX Programmer's Manual. Starting with a new disk layout is highly recommended.

Second, **re-use existing descriptive file**, where appropriate, such as */dev, /dev/ttys, conf.d, /etc/rc, header.s*, if you are upgrading from an existing customized system. However, changes for standardization may be necessary in */dev*, since the major device numbers should agree with Table 7.2 below and the kernel processes may have to be renamed in */prc*. From your old source, you also like to have available the driver source if it has nonstandard device addresses. Find out where they are, so that you can compare them with the distributed source. (The old source should not be mounted on */src*, but on some other mount point, e.g. */crp*).

Third, **prepare to read the Source Tape**. You need the binary of the *cpio* command and a shell procedure *extract.sh* containing invocations of *cpio* to read the Source Tape. You must obtain these two files from the root file system on the File System Tape. If you're coming up from scratch (Section 19), you will already have mounted this file system on *src*; so skip this paragraph. However, if you are not running on the distributed root system, you must extract the source from the supplied Source Tape, which is in *cpio* format. You don't have the *cpio* command though, and the extract shell using *cpio* for reading in the Source Tape onto your system. To acquire these two pieces from the distributed root file system, mount the File System Tape, then copy it into some free file system, and mount it on a mountpoint, e.g. */mnt*, and copy the *cpio* and *extract.sh* files:

```
dd if=/dev/mto of=/dev/rp2 count=4000 skip=100
/etc/mount /dev/rp2 /mnt
cp /mnt/bin/cpio /bin
cp /mnt/bin/extract.sh /bin
chmod~~766~~/bin/cpio~~/bin/extract.sh
```

Fourth, **transfer source from tape to disk**, by invoking the shell command:

```
extract.sh all
```

which will read the whole Source Tape into the structure under *src*. Follow this with a *du -a* command, to make sure none of the files read has zero length. Reread missed files using *cpio*-I. If you are an RK-only installation, follow the instructions given in Section 19, under 9.*The Source Distribution Tape*, to extract smaller portions from the tape. The */src* substructure will

now contain the following directories:

| | |
|---|---|
| mertsrc | MERT operating system source |
| cmd | user command source |
| lib | library source |
| mdec | boot programs source |
| makefile | shell scripts to remake cmd |

Familiarize yourself with this organization of the source.

Fifth, **recreate the C compiler and associated libraries**. The C language will probably not undergo major changes in the future. More likely, the C compiler will be getting improved continually, e. g. the code generation. Newly distributed software will take advantage of the changes, and, more crucially, will have been tested only with the latest compiler. It is therefore essential, that you recreate the compiler for your system very early. This is mandatory, if you have no floating point hardware on your system, since the distributed software, (except for the bootable distribution system), assumes the presence of floating point hardware. Since it cannot be assumed that the newly distributed compiler is compilable with the old compiler, you have to obtain a binary copy of the new compiler from the distributed root system. Proceed as under 'Third' above to get the 'root' file system mounted on */mnt*, then copy the appropriate binary files, e. g.:

```
cp /mnt/bin/cc /bin/cc
cp~~/mnt/...~~/bin/...
```

Now you execute one of the *make* files you find in */src/cmd/c*.

The subsequent paragraphs describe various parts of the source which may require editing. Read through these all; in paragraph 22. *sgen* you will finally encounter the command to generate the whole operating system.

If all the source has been edited etc., it would suffice to use the shell command *makemert* which aids in the remaking of the MERT operating system. However it is not recommended to do this the very first time. Rather, you should go through these procedures step by step and try to understand what is going on. After you have gone through it successfully once, you may use the abbreviated procedure.

In the source structure mounted under */src*, you will find, besides the source of MERT and UNIX, a multitude of system generation aids explained in the next paragraph. These have the advantage of using a new program called *make*, which can save you considerable time when repeatedly compiling any source.

## 2. Generation Aids and General Formats

The discussion below assumes you have booted up the appropriate MERT system from your root file system disk. It assumes you have formed a C compiler with or without floating point as appropriate for your machine. The MERT source file system is assumed to be mounted on */src/mertsrc*.

Each subdirectory of */src/mertsrc* contains three shell scripts which can be used for recompiling all the elements within the directory. Two of the shell scripts use the *make-I* command and will only remake an element if it is out of date. The third shell script unconditionally remakes every element within a directory. One make shell script assumes source dependencies and the other make shell script assumes object dependencies. To distinguish between the three, the following naming conventions have been adopted:

1. make*[os].sh
2. *[os].mk
3. run

1. is the general form of a make shell name; 2. is the general form of a make descriptor file; and 3. is the name of the shell that recompiles every element within a directory. Each make shell begins with the word *make*. The '*' is usually the first four characters of the directory name. The 'o' or 's' signifies a make shell object or source dependency. As an example:

makebdevo.sh
tuprco.mk

The file makebdevo.sh is the name of a make shell which would remake any block or record device process in *bdev* that is not up-to-date. The shell assumes that all the corresponding *.o files are in the directory. If not, they are remade. The file tuprc.mk is a make descriptor for the process *tu* (the TU16 device driver). This file lists the dependent files of the target *tuprc*.

The make shells often reference the run shell scripts when it has been determined that a element within a directory has to be remade. The run shell scripts contain the various commands that remake an item. These run shell scripts and make files should be consulted before you remake any item. The procedural steps below will frequently refer to run shell scripts. When remaking an item use the make shell scripts instead of the run shell scripts. They have the advantage of speed and dependency accuracy. They won't compile everything in sight, and if module x, y, and z depend on module a, and module r, s, and t do not, and module a has changed, then only module x, y, and z will be remade.

To further aid in the generation process, a shell called *makemert* has been provided. The command:

makemert run [filemanager]

recompiles every directory under /src/mertsrc unconditionally. However, this takes approximately one hour on the 11/70 without doing a *sysgen*. Replacing *filemanager* with the string *unix* or *mert* recompiles the appropriate filemanager. Remember that the system is delivered with a MERT file manager. Some installations may prefer to run MERT with a UNIX file manager. This allows you to run both the UNIX and MERT operating systems on the same file system. See paragraphs 16. and 17. below for instructions.

The command:

makemert mko [filemanager]

conditionally recompiles each directory under */src/mertsrc*, using the object-dependent make files. It will also perform a sysgen and squirrel away the old *unix, mert*, and *krn.sym*. The command:

makemert mks [filemanager]

performs the same as *mko* above but uses the source-dependent make shells.

If you only wish to regenerate the operating system and not every directory under *mertsrc*, then in the directory *sgen* you will find two make shells that have been provided for this purpose. They are called *makemerto.sh* and *makemerts.sh*.

To generate a completely new system, the following procedures should be followed. If this is your first attempt at 'sysgening' a system, follow these steps closely. Most steps may be skipped if you already have a running system and just want to add a few new drivers.

**3. conf.d**

The 'conf.d' file contains definitions of how many logical channels of each device are to be included in your system. Edit it according to your configuration. The 'BASE' definitions should not be altered without recompiling all source files which include 'conf.d'. In particular, *kfs/init.c*, *kfs/ps.c* and *kfs/tkill.c* should be recompiled along with any character device drivers affected. *Init.c, ps.c,* and *tkill.c* can be made by running *makekfs.sh* or *makekfso.sh.*

**4. kfs**

This directory contains the source for many user programs which differ from UNIX source as well as all of the file system utility programs. The latter are contained in two subdirectories: *mkfs* contains the utilities and programs that go with the MERT file manager (see 15. below), whereas *ukfs* contains utilities and programs that go with the UNIX file manager (see 16. below). Each of these subdirectories contains the shell files for making as well as installing the respective commands. Switching from one file manager to another requires remaking of the associated subdirectory (*mkfs* or *ukfs*).

The 'run' file contains the shell script for recompiling all of the source code in this directory.

Note that the file 'ttyc.c' has a number of tables which contain the list of the 'tty' device file names for the various devices which can be used as login terminals. The number of entries in each table should be the number of logical channels allowed on the corresponding device. For example, if NDH11 is defined as 10 in 'conf.d', there should be ten characters in the 'ndh11' table structure. Both the 'ps' and the 'init' commands include this 'ttyc.c' file when '/bin/ps' and '/etc/init' are compiled, respectively.

The 'init.c' file contains another important table which should be looked at carefully before compiling. This table contains a list of special 'getty' routines to be invoked when a particular login terminal is started up. Make sure that the entries in this table are complete for your system. Also make sure that the order of the entries in this table corresponds to the entries in the '/etc/ttys' file. This file contains the list of all the legitimate login devices which may start up a user process. The file is accessed by the '/etc/init' program. Each entry in */etc/ttys* consists of three characters followed by a newline. The first character is either 0 or 1, 1 if the device is to be a valid login device. The second character is the last character in the name of a typewriter, e.g. 'a' refers to '/dev/ttya'. The third character is used as an argument to the '/etc/init' program to decide which '/etc/getty' to execute to read the login name, set the baud rate and certain default terminal options. For most lines, the third character is '0' indicating '/etc/getty', the first entry in the table in 'init.c' is to be executed. For example, if the third character of an entry is '2', the second entry in the table after '/etc/getty' is used. See ttys-V for further details.

There are a number of programs which should be changed in this directory to reflect your default mounted file system devices. The first few lines of these programs (*df, check, dcheck, icheck* and *ncheck*) contain a list of these devices and should be edited accordingly. In addition, the reconfiguration daemon *recdmn* source code should be edited so that the proper file systems are reconfigured automatically once a day. See *recdmn-d* for a complete description of this program.

To compile the other user programs consult the 'run' file for the appropriate shell scripts. The 'ls' and 'mknod' programs are included here because they have been modified to recognize record-type devices.

**5. elib**

In this directory, the shell script in 'run' should be followed to form the library routines which interface the supervisor EMT's to the kernel in the library archive file */lib/libe.a.*

**6. klib**

In this directory, the shell script in 'run' should be followed to form the library routines which interface the kernel process EMT's to the kernel in the library archive file */lib/libk.a.*

**7. rlib**

In this directory, the shell script in 'run' should be followed to form the library routines which interface the new MERT-UNIX system calls to the MERT-UNIX supervisor in the library archive file */lib/libr.a.*

**8. slib**

In this directory, the shell script in 'run' should be followed to form the general supervisor routines used by supervisor processes in the library archive file */lib/libs.a.*

**9. adm**

In the 'adm' directory, consult the 'run' file to recompile all of the maintenance and utility programs. The shell scripts described in the 'run' file will form the following programs: *adb, sdb, kdb, cdb, ldp, ldu, sgen* and *xusr.*

**10. appl**

This directory contains some application and utility programs as well as some test programs for the new MERT-UNIX system calls (those not existing under UNIX). The application programs which should be compiled include *acp, errproc, falloc, fmove, kdmp, ktime, pcp, pio, run* and *tp.* The *tp* command differs from UNIX by having additional asynchronous I/O capabilities. Refer to the MERT Programmer's Manual for the use of the other MERT-UNIX commands. Consult the 'run' file for instructions on recompiling the source for the commands.

**11. System Library**

The time of creation of the system library is very important in the generation of a new MERT system. Consult *cdev/libsh* for the shell sequence required to form a new system library. The library is generally put in */mrt/syslib* and is used by character device drivers which require the system library routines. It is loaded in the bootable image of MERT. It is important that the version loaded here is the same version that is used by the character device drivers; otherwise these character device drivers will fail to load when the system is booted up. Note: if you make a new library and use it in the boot image, all character device drivers which reference the system library must be recreated.

**12. cdev**

Character devices on the system must have entries in directory */dev* (see 7.*Special Files*, in Section 19), and in file */etc/ttys* if they are used for remote access (see 8.*Multiple Users* in Section 19). Character devices are also affected by entries in global source files *conf.d* (see 3. above), *kfs/init.c*, and *kfs/tty.c* (see 4. above).

The specific source for the individual devices is contained in directory *cdev* including the following:

dc DC11 asynchronous communication device

dh DH11 programmable asynchronous communication device
dl DL11 serial asynchronous device for Satellite Processor System
dm DM11 multiplexed asynchronous device
dn DN11 automatic calling unit
dp DP11 synchronous communication device
dr DR11 general device interface
du DU11 synchronous communication device
kl KL11 asynchronous console interface
lp LP11 line printer
mem memory device driver
sdh DH11 programmable asynchronous communication device for Satellite Processor System

For each driver there is a shell script file which should be consulted to form the appropriate driver process file. For example, for the console device driver, the kl, there are four files of interest:

| | |
|---|---|
| kl.c | source code |
| klmch.s | machine language assist |
| klsh | shell script |
| kl.b | specification file. |

In general, for each driver one should check that the interrupt vector(s) and the device register address are defined correctly in the source code for the driver. Note also that these must correspond to the interrupt vector(s) and device address specified in the 'sys.b' file in the 'sgen' directory, as described in a later section. If this is not true, one will not be able to load the device driver at all since the attach interrupt EMT will fail. The 'klsh' file contains:

```
cc -c -O kl.c
as klmch.s;mv a.out klmch.o
ldp kl.b
```

The 'kl.b' specification file contains parameters to be used by the 'ldp' utility program:

```
mode:       kl
interrupt:
{
            rconsol,klrint0
            xconsole,klxint0
}
emt:        _klemt
event:      _klevent
ifile:      kl.o klmch.o
ofile:      klprc
idchar:     K
```

The various keywords are explained in detail in *ldp-e* in the MERT Programmer's Manual. The *mode* keyword here indicates that the kl driver is a kernel mode process and is to include the system library. The *interrupt* keyword indicates the names of interrupt vectors which are defined in 'klmch.s'. The *emt* and *event* keywords specify the emt and event entry points, respectively. The *ifile* keyword indicates which files are to be link-edited to form the image of the kl console kernel device driver. The *ofile* keyword indicates where the image of the driver is to be loaded. This is a crucial specification. The process image file is created in the directory. There is a shell called *move2process*, that copies the file image to the appropriate /prc/cdX. Check that this shell is correct for your system. There is an important connection

between the process image in the '/prc' directory and the file system name as listed in the '/dev' directory. See **7. Special Files** in Section 19, for a further explanation. Note that the console device driver is always loaded as '/prc/cd0' ( the first character device driver). The *idchar* keyword specifies the one-character code by which the driver is known in the file '/mrt/kprc'. See *ps-e* in the MERT Programmer's Manual for a further explanation of this connection. When the process image is loaded in the *ofile* file, the library '/lib/libk.a' is searched for any undefined routines.

Finally, here are some hints on some of the drivers. There are certain magic numbers and configuration parameters imbedded in various device drivers that you may want to change. The device addresses of each device are defined in each driver. In case you have any non-standard device addresses, just change the address and recompile.

The DC11 driver is set to run 14 lines. This can be changed in 'conf.d'.

The DH11 driver will only handle a single DH with a full complement of 16 lines. If you have less, you may want to edit 'conf.d'.

The DN11 driver will handle 3 DN's. Edit dn.c.

The DP11 and DU11 drivers can only handle a single DP and DU, respectively. This cannot be easily changed.

The KL/DL driver is set up to run a single DL11-A, -B, or -C (the console) and no DL11-E's. To change this, edit 'conf.d' to have NKL11 reflect the total number of DL11-ABC's. So far as the driver is concerned, the difference between the devices is their control and status register addresses ( which are assumed to be consecutive).

The line printer driver is set up to print the 96 character set on 80 column paper (LP11-H) with indenting. Edit lp.c.

### 13. bdev

This directory contains all of the source for the block and record device drivers. This includes the following:

| | |
|---|---|
| hp | RP04 disk drive |
| hs | RS04 disk drive |
| ht | TU16 tape drive |
| rf | RF11 fixed head disk drive |
| rk | RK05 disk cartridge |
| rp | RP03 disk drive |
| tc | TC11 DECtape |
| tf | Telefile disk drive |
| tm | TU10 tape drive. |

For each driver there is a shell script file which should be consulted to form the appropriate driver process file. For example, for the RK05 device driver, the *rk*, there are three files of interest:

| | |
|---|---|
| rkprc.c | source code |
| rkprc.b | specification file |
| rksh | shell script. |

One should check that the device register and interrupt vector addresses are defined correctly in the source code for each driver on your system. Note also that the interrupt vector address must correspond to the address specified in the 'sys.b' file in the 'sgen' directory, as described in a later section. If this is not true, one will not be able to load the device driver at all since

the 'attach interrupt' EMT will fail when an attempt is made to load the device driver. The 'rksh' file contains:

```
cc -c -O rkprc.c
ldp rkprc.b
```

The 'rkprc.b' specification file contains parameters to be used by the 'ldp' utility program:

```
mode:       k
open:
ifile:      rkprc.o
ofile:      rkprc
interrupt:  rk11,_rkintr
event:      _rkstrategy
swap:       4000.,872.
ubmap: 8
```

The various keywords are explained in *ldp-e* in the MERT Programmer's Manual. Most of the keywords used above are discussed in the preceding paragraph on the character device drivers. Note that this is a kernel-mode device driver which does not use the system library. The interrupt entry point is given as '_rkintr' which is defined in 'rkprc.c'. The hardware priority is given by the key 'rk11' which is a DEC standard device defined in the 'ldp' program. The event entry point is entered at priority 5 as defined by the device hardware priority by the 'rk11' key. The *open* keyword indicates that the driver accepts open and close messages. This is an important keyword for drivers that are loaded dynamically since it assures that all outstanding I/O messages are serviced before the final 'close' message is acknowledged and the driver process is removed from memory. If the driver process is to be a part of the 'boot' image, 'open' and 'close' messages are not necessary. Then the code to deal with these messages may be (but should not be) commented out of the driver. The *swap* keyword is necessary if a swap area is to be allocated on the disk. Similarly, the *ubmap* keyword is used to define how many 4K words sections of the UNIBUS map on the PDP-11/70 must be allocated for this device. It is only required for UNIBUS devices on the PDP-11/70 computer but not for MASSBUS devices. Again the *ofile* keyword is crucial here. As in the *cdev* directory, check that the *move2process* shell is correct for your system, since there is an important connection between the process image in the '/prc' directory and the major device number of the device as listed in the '/dev' directory. When the process image is loaded in the *ofile* file, the library '/lib/libk.a' is searched for any undefined routines.

All of the disk and tape drivers (rfprc.c, rkprc.c, rpprc.c, tmprc.c, tcprc.c, hsprc.c, hpprc.c, htprc.c, tfprc.c) are set up to run 8 drives and should not need to be changed. The big disk drivers (rpprc.c and hpprc.c) have partition tables in them which you may want to experiment with. However, they comply with the documentation in *RP-IV* and *HP-IV*.

In summary, for each block and character device driver make sure that the interrupt vector(s) and the device register addresses as specified in the source code for the driver correspond to entries in the 'sgen/sys.b' file for your system. Also make sure the *ofile* keyword for each of the drivers you wish to load is correct. Then you may run the corresponding shell scripts. Note that the correct device names with the appropriate major device numbers and the correct type are created in the '/dev' directory using '/etc/mknod'. The only record-type devices are the two magtape drivers. The others are all block-type devices.

**14. pmge**

This directory contains the source for three supervisor processes. One need only to run the three shell scripts in *nubsh, pkillsh* and *pmsh* to form the file images of these three processes. The image of the 'nub' process which starts up all other supervisor-user processes is created in 'sgen/nubprc'. The image of the 'pmgr' process is created in 'sgen/pmprc'. The image of the 'pkill' process is created in '/prc/pkill'. For all supervisor processes, the library '/lib/libs.a' is searched for any undefined routines by the 'ldp' program.

**15. fmgr: MERT File Manager**

As a default, the system will be created with the MERT file manager. Optionally, a UNIX file manager may be created instead, as explained below under 16.

If you are running with a UNIX file manager and want to switch to a MERT file manager, remake directories */src/fmgr* and */src/kfs*, in that order. Performing the latter will install the changed file manager in the *header.s* file and remake and install the affected commands. Instructions for creating the MERT file manager process in 'sgen/fmprc' are contained in the 'run' file in this directory. The 'shc' file contains the script for compiling all of the modules which make up the file manager process. There are two parameters in 'param.d' which one may wish to change. The NBUF parameter is set at 10 and the NMOUNT parameter is set to 4. If more mounted file systems are required, the NBUF and NMOUNT parameters may be increased. Make sure that all of the modules which include these two parameters are recompiled. The number of tasks which the file manager can handle is set at 8. If more are required, NTASKS must be changed in 'param.d' as well as making some corresponding changes in 'fmgr.s'. It is doubtful that this value should be increased, however. To produce the file image of the file manager process, run the 'shl' script file.

**16. unixfmgr: UNIX File Manager and ukfs directory**

If you are running with a MERT file manager and want to switch to a UNIX file manager, remake directories */src/unixfmgr* and */src/kfs/ukfs*, in that order. Performing the latter will install the changed file manager in the *header.s* file and also make and install the appropriate utilities *ncheck, dcheck,* etc. The *ukfs* directory contains counterparts of all programs in *kfs* that are file-system dependent, including shell and makefiles for generation and installation. The comments above under 15. about the number of buffers and number of mounted file systems also apply.

**17. UNIX Supervisor**

The various modules which make up the UNIX supervisor process can be compiled by running the 'shc' script file:

```
cc -c -O *.c
as mch.s;mv a.out mch.o
cc -c sys4.c
```

The 'sys4.c' file must be compiled separately without the optimizer, since the C compiler optimizer does not treat the transfer within a routine correctly for the 'sys fork' system call. To increase the number of system buffers available to all UNIX users, increase the value of NBUF in 'param.d' (normally set to 8). The upper limit for this parameter is probably about 24. To produce the final file image of the UNIX supervisor process, run the 'sht' script file:

```
ldp unix.b
```

This produces the process image in the file 'test'. To ensure that this is a proper supervisor process, one may 'run test' (see *run-e*) and 'login' to the new supervisor to perform some tests.

This is the recommended procedure for testing out a new UNIX supervisor when modified or new system calls are installed. When one is satisfied that the UNIX supervisor is working properly, perform the following commands as superuser:

```
cp test /prc/nunix
chdir /prc
mv unix ounix;mv nunix unix;sync
```

This ensures that the new UNIX supervisor is installed properly, saving the old process image in case of problems. A simple copy of the new supervisor file to the old supervisor file may well create havoc, since the common text and data segments in the supervisor would not be loaded since their segment names would correspond to those of the old segments. Note that 'mv'ing a file does not change the names of the segments making up a process.

A reboot of the system is not necessary to install a new UNIX; however it is recommended at some time, to avoid the loading of two UNIX supervisor processes.

### 18. Commands

The command source is contained under */src/cmd.* Single file commands are file entries in this directory, multifile commands such as *yacc* are subdirectories. Aids to remake commands are contained in */src/makefile.*

### 19. Time Zone Conversion

If your machine is not in the Eastern time zone, you must edit (ed-I) the subroutine /src/lib/libc/clib1/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. This routine also contains the names of the standard and Daylight Savings time zone; so 'EST' and 'EDT' might be changed to 'PST' and 'PDT' respectively. Notice that these two names are in upper case and escapes may be needed (tty-IV). Finally, there is a 'daylight' flag; when it is 1 it causes the time to shift to Daylight Savings automatically between the last Sundays in April and October (or other algorithms in 1974 and 1975). Normally this will not have to be reset. After ctime.c has been edited it should be compiled and installed in its library. (See /src/lib/libc/makefile) Then you should (at your leisure) recompile and reinstall all programs performing time conversion. These include: (in /src/cmd) date, dump, ls, cron, mail, pr, restor, who, sa and tp.

### 20. header.s

This file in the 'sgen' directory contains certain option settings which are important for your installation (a '0' means 'no', a '1' means 'yes' in answer to the statement or question at right.)

```
fpp =0      /hardware floating-point processor in system?
pfs =1      /power fail safe recovery routine?
debug =0    /kernel debugging mode on?
.hp =0      /produce core dump on rp04 (hp04)
.rp =0      /produce core dump on rp03
.tf =0      /produce core dump on telefile disk
.tc =0      /produce core dump on DECtape
.rk =1      / produce core dump on rk disk
```

This file is included when assembling various parts of the system in the 'kern' directory.

MERT supports the 11/45 FP11-B and the FP11-C **floating point processors**. The operating system as delivered has *fpp* set to 0 and therefore has been generated for non-floating point

hardware (this is so the system will boot on non-floating point machines). The 'fpp' flag should be set to 1 if your system contains a hardware floating-point processor, otherwise to 0. However, if you run a system with a floating-point unit, have fpp = 0 and come up multi-user, programs using the floating-point unit may not produce correct results, since the floating-point unit registers will not be saved in a context change. You should therefore regenerate the system with *fpp = 1*. The commands that come with the system have been generated for floating point machines. If you do not have floating point hardware, then the following directories will have to recompiled:

/src/lib/liba
/src/lib/libc
/src/cmd/cc
/src/cmd/sccs
/src/mertsrc/adm/adb

The **power fail safe** option may be included by setting the flag 'pfs' to 1. This ensures that all volatile registers are saved in memory when power goes down and that each process in the system receives an INIT message when power comes back on.

The **debug** flag is normally set to 0. It should only be set to 1 when one wishes to debug the kernel by causing the system to crash when certain detected conditions occur. It should also be set to 1 if one wishes to collect statistics with the kernel monitor process (see below under *23. mon* ), and if you wish an EMT histogram from the *ktime* command.

Of the remaining five flags, only one should be set to 1, the others to 0. The flag which is set to 1 (one of .hp, .rp, .tf, .tc and .rk) defines on which device a dump of memory is produced when the system crashes. Check the 'kern/panic.s' file for the area on disk used for the dump in each case. It is important that this area not overlap with any file system which may exist on the disk. It may also be necessary to change the drive number (*drvno*) if you wish dumps to occur on a drive different from drive 0.

**21. kern**

This directory contains the source code for the kernel and the 'init' process which loads all of the startup processes at boot time. There are a number of shell scripts for forming the various pieces. The 'run' file contains the complete script:

```
as -o panic.o ../sgen/header.s kern00.s panic.s
sh shs
as -o kern.o ../sgen/header.s kern*.s
sh shl
sh shi
```

The first line of this file assembles the panic routine into 'panic.o'. The second line assembles and compiles various pieces of the scheduler. Most of the scheduler is written in C. The third line assembles the other parts of the kernel into 'kern.o'. Finally all of the pieces of the kernel are link-edited together by 'shl' producing the final output file in 'sgen/kern.o'. The initialization process is assembled by 'shi' producing the image of the process in 'sgen/init.o'. Note the use of the 'header.s' file in many of these shell scripts. If a change is made in 'header.s', make sure that all of the shell scripts which specify 'header.s' are run again.

**22. sgen**

This directory contains all of the object modules which make up the boot image of the MERT Operating System:

| | |
|---|---|
| fmprc | file manager process |
| init.o | initialization process |
| kern.o | kernel (includes scheduler and memory manager) |
| lcor.o | low core image |
| nubprc | nub process |
| pmprc | process manager |

The other important files in this directory include the 'sys.b' specification file which contains lines consisting of a keyword followed by a parameter list:

```
kernel      kern.o
lowcore     lcor.o
swapprc     /prc/bd0
rootprc     /prc/bd0
fmgr        fmprc
pmgr        pmprc
init        init.o
syslib      /mrt/syslib
nub         nubprc
user        /prc/cd0
user        /prc/unix
swapdev     0 0
rootdev     0 0
messages    32.
processes   50.
ports       2
memory      0,64.
nrsde       180.
console
rk11
rp11
hp11
tf11
tc11
tm11
tu11
stack       768.
v 314,170500  /* dm11-bb - non-standard device ordering
v 320,160020  /* dh11 receiver - non-standard device ordering
v 324,160020  /* dh11 transmitter - non-standard device ordering
```

The keywords and the allowable parameters are discussed in detail in *sgen-e.* Of the six object modules listed previously, five were generated from other directories. The 'lcor.o' file is produced by 'sgen' itself from the assembly language file 'lcor0.s' and 'lcor1.s'. Of these, 'lcor0.s' is generated by the 'sgen' program from the low core vectors specified in 'sys.b' and from the other keywords which affect low core, such as number of message buffers, number of resident segment descriptor entries, etc. The 'lcor1.s' file contains constants which must be in I = D = physical memory.

Both root and swap processes are included in the boot image. They are normally the same. The system library is almost always included since most login character device drivers include references to the system library. Note that the system library is kept in the '/mrt' directory. The version of the system library used must be the same as that used by the character device drivers being used. The *rootdev* and *swapdev* keywords specify the major and minor device numbers of the root file system and the swap area, respectively.

The *user* keyword is used to specify which processes are started up by the kernel initialization process at boot time. Normally the console device driver and the UNIX supervisor processes are started up. If your system has a DH controller,the corresponding character device process should also be started up at boot time.

The last keywords in the 'sys.b' file starting with 'console' specify devices which have interrupt addresses and control and status registers which must be specified. All of the devices which are connected to your system, whether or not the driver process is part of the boot image, must be specified here. The interrupt vector address for each device must correspond to that specified in the driver's specification file in the 'bdev' or 'cdev' directory. Otherwise that particular device driver process cannot be loaded at run time. The first keywords correspond to standard DEC devices and are "built-in" *v* keywords. An example of a non-standard vector is shown for the DH device in the last three lines of the 'sys.b' file. Extraneous keywords in this list may be removed.

Having verified that the 'sys.b' file is correct for your configuration, you write

```
sgen sys.b mert
```

to produce a bootable core image in 'mert'. Note that a symbol table for the kernel is also produced in 'krn.sym' in the 'sgen' directory. The symbol table corresponding to the currently running system should be kept in '/mrt/krn.sym' since it is used by the process status (*ps-e*) and the kernel debugger (*kdb-e*) programs. To test out the new MERT boot image:

```
cp mert /tmert
sync
```

Then halt the system and boot up */tmert*. If the system crashes, bring up */mert* and use 'kdmp' to produce a core image of the crash in 'kore' in the 'sgen' directory. Use the kernel debugger 'kdb' to debug the dump. When you are confident that the system is working:

```
mv /tmert /mert
cp krn.sym /mrt/krn.sym
```

Note that the shell makemert[os].sh does most of the above for you.

**23. mon**

This directory contains source code for a kernel monitor process. The process collects scheduling information and puts it in a file 'profile'. The 'log' command in this directory can be used to examine the output of the monitor. Read the 'roff_me' file for instructions on how to compile and load the process, how to collect statistics and how to examine the output.

**24. publib**

This directory contains some examples of the use of public libraries for UNIX user programs. Examine the 'run' shell script file for an example of putting together a program which includes two libraries.