3

MINIX ON THE ATARI ST

In this chapter we will describe how to boot and install MINIX on the Atari ST. It is assumed that the reader is already familiar with MINIX in general, and has at least some knowledge of UNIX.

Booting and installing MINIX on the Atari ST is complicated by a variety of factors:

- Atari STs are sold with a wide variety of incompatible keyboards
- Some versions can only handle 360K diskettes; others can handle 360K and 720K diskettes
- Some people have winchester disks (hard disks); others do not
- The amount of memory available ranges from 512K to 4M
- The Mega ST differs in some ways from the Atari ST 520 and 1040

Together, these different configurations give problems. Our solution has been to provide a base version that will require at least 1MB of memory and one 720K disk drive, and make it possible for people with larger configurations to adapt the system to take advantage of their extra hardware. This chapter explains how that is done.

It is possible to run MINIX on a system with 512K of memory, but that leaves very little space for applications. In this case the best thing to do is to work without a

ram disk at all, and keep the root filesystem on either hard disk or diskette. Running MINIX on a system with only a 360K disk drive is also possible. However in that case you must first split each 720K diskettes in the distribution into two 360K ones. Since you do not have a disk drive capable of dealing with 720K diskettes, you should do this on a friends system. Sec. 3.6 describes how to split a 720K diskette into two 360K ones.

3.1. THE MINIX-ST DISTRIBUTION

The MINIX-ST distribution consists of ten diskettes. One of them contains a binary of the operating system and is used for booting MINIX-ST. Eight others contain MINIX file systems. Only one of them contains a TOS file system. (We use TOS as the name for any combination of BIOS, XBIOS, GEMDOS, GEM, AES and VDI).

All distribution diskettes are double-side, and formatted on both sides. However, two diskettes contain only 360K of information written on one side of the diskette. In other words, these two diskettes are written as if they were single sided diskettes. Here is the list of the diskettes:

Name	Sides	Size	File sys.	Description
00.TOS	DS	720K	TOS	utilities that run as TOS programs
01.BOOT	DS	360K	special	used for booting MINIX
02.ROOT	DS	360K	MINIX	root file system copied to RAM disk
03.USR1	DS	720K	MINIX	most commonly used commands
04.USR2	DS	720K	MINIX	commands part 2 of 3
05.USR3	DS	720K	MINIX	commands part 3 of 3
06.ACK	DS	720K	MINIX	compiler binaries and libraries
07.SRC1	DS	720K	MINIX	sources of the MINIX operating system
08.SRC2	DS	720K	MINIX	sources of commands part 1 of 2
09.SRC3	DS	720K	MINIX	sources of commands part 2 of 2

We will refer to these diskettes in the rest of this manual by their name in the first column of this table, for example, 01.BOOT.

Before you start working with these diskettes we urge you to copy all of them. You can use normal TOS procedures, like dragging icon FLOPPY DISK A onto icon FLOPPY DISK B, to make the copies. Whenever we refer to diskettes as 01.BOOT, 02.ROOT, 03.USR1 etc. we always mean a write-enabled copy of the original diskettes. Store the original diskettes after copying and keep them write protected

under all circumstances. Do not use your originals as work diskettes.

3.2. NATIONAL KEYBOARDS

The Atari ST comes with different keyboards in different countries. This lack of standardization is a major nuisance. Atari solved the problem by providing a different version of their operating system for each country. We have chosen a different strategy: a single version that can be adapted to the various keyboards. This section describes how to set up MINIX for your keyboard.

Unless you have an Atari ST with a United States version of the keyboard, you *must* first adapt MINIX to your particular version of the keyboard. Even with a United States version this procedure can do no harm, so if in doubt proceed. If you skip this procedure, it is assumed that the keys generate the characters that are engraved on the key tops of the United States keyboard, that is, the key below the DEL will generate the ASCII backslash character (\) unshifted, and the ASCII bar (|) if shifted, irrespective of the character engraved on the key tops of your keyboard.

MINIX cannot handle the national characters themselves, like o-umlaut for Germany. The adaptations described below only allow you to enter the ASCII characters in the way you are used to with TOS. In this respect MINIX behaves like most versions of UNIX.

MINIX has its own keyboard translation tables build into the operating system. A special tool is provided to extract the keyboard tables from a running version of TOS and to adapt the tables in the binary version of the MINIX kernel accordingly, without the need to recompile the MINIX kernel. Note that the MINIX keyboard translation tables have exactly the same format as used by TOS.

Some keyboard versions need so many keys for special non-ASCII characters that combinations with the Alternate (ALT) key are used to generate some ASCII characters. For instance, in France the key below the Delete (DEL) generates the ASCII sharp sign (#), SHIFT-# generates the bar (|), ALT-# generates the at-sign (@), and ALT-SHIFT-# generates the tilde. The keyboard tables do not take the ALT key into account, so Atari delivers several national versions of the TOS operating system to cope with these problems, as mentioned above. In order to avoid national versions of MINIX, we have built into the keyboard driver a little table of special ALT and ALT-SHIFT combinations for the limited number of national keyboard versions that we knew of: United States, United Kingdom, Germany, France and Spain. If you happen to have another version you can make a simple modification in the keyboard driver of the kernel, but that takes effect only after recompiling the kernel. Refer to the chapter on kernel recompilation.

To adapt the keyboard tables proceed as follows:

1. Boot TOS and insert 00.TOS in drive 0.
2. Open a window onto drive 0 by double clicking the FLOPPY DISK A icon.
3. Run *COMMAND.TOS* found on diskette 00.TOS by double clicking. *COMMAND.TOS* is a simple line-oriented command interpreter.
4. Run *FIXKEYS.PRG* by typing

   fixkeys a:

5. Insert unprotected 01.BOOT
   when you are asked to do so, and confirm by hitting the RETURN key.
6. Wait for the program to reply with

   Done

You are now ready to boot MINIX.

3.3. BOOTING MINIX-ST

This section presents a boot procedure for MINIX-ST that works on all configurations of the Atari ST. Following sections describe how to adapt the set of diskettes so that you can use MINIX effectively on your particular combination of memory and disk drives. For example, if you have more than 512K you can increase the size of the RAM disk from 160K to 300K, if you have 1M of memory, or to 1M or more if you have even more memory. If you have a hard disk, all of the diskettes can be copied onto one or more of its partitions. Finally, some of the options for booting MINIX will be explained. But first the procedure for booting that works on all configurations is described.

Throughout the discussion below, lines printed in the Helvetica typeface are either commands you should type on the keyboard, or are lines that the computer will display for you. In a few of the examples, *italics* characters or words appear in a command. These represent values that you are to fill in.

Booting is a three stage procedure. First the operating system itself is loaded into memory. Then the ROOT file system is copied to a RAM disk allocated in memory. Finally, the script */etc/rc* is executed and a message will be displayed on the screen asking you to log on.

To boot MINIX-ST, proceed as follows:

1. Turn off the ST and then insert diskette 01.BOOT in drive 0. You could push the RESET button as well, but that may not free memory occupied by a crash resistant TOS RAM disk you may have running. Moreover, it fails if you normally boot from the winchester.

2. Wait ten seconds, then turn on the ST. It will read the operating system (about 153K) from diskette in a few seconds. The screen will turn black and it will show on the top two lines the message:

```
Booting MINIX 1.5.  Copyright 1990 Prentice-Hall, Inc.
Insert ROOT diskette and hit RETURN (or specify bootdev)
```

3. Replace 01.BOOT by diskette 02.ROOT and hit RETURN. Alternatives will be explained later. The system will respond with:

```
Memory size=992K  MINIX=153K  RAM disk=160K
Available=679K
```

   for a system with 1M of RAM (numbers might deviate a little). Adding 32K (the size of the video memory) to the first number should give the amount of memory in your ST.

4. A fourth line will be displayed that reads:

```
RAM disk. To load:  120K          Loaded:  0K
```

   (The number 120 may vary a little). In rapid succession the number 0 will be increased in steps of 18K, until the whole line is replaced by:

```
RAM disk loaded.    Please remove root diskette.
```

5. When the RAM disk is loaded, the system initialization file, */etc/rc*, is executed. It asks you to remove the root file system and insert the */usr* file system (03.USR1) in drive 0 and type a RETURN. Do so.

6. After */usr* has been mounted, you will next be requested to enter the date (and time). Enter a 12-digit number in the form MMDDYYhhmmss, followed by a RETURN. For example, 9:35 p.m. on June 01, 1990 was 060190213500.

7. You will now get the message:

```
login:
```

   on the screen. Type:

```
root
```

   and wait for the system to ask for your password. Then type:

```
Geheim
```

being careful to type the first letter in upper case. Lower and upper case letters are always distinct in MINIX. Alternatively, you could have used the name ‘‘ast’’ together with the password ‘‘Wachtwoord’’. This is much preferred when you use the system normally, but for now it is troublesome.

8. If you have successfully logged in, the shell will display a prompt (sharp sign for root, dollar sign otherwise) on the screen. Try typing:

   ```
   ls –l
   ```

   to see what is in the root directory. Note that you need six keystrokes: ‘‘l’’, ‘‘s’’, space, ‘‘–’’, ‘‘l’’, and a RETURN. Then type

   ```
   ls –l /bin
   ```

   to see what is in the */bin* directory on the root device (RAM disk). After that, try:

   ```
   ls –l /usr/bin
   ```

   to see what is on the drive 0 diskette. To stop the display from scrolling out of view, type CTRL-S; to restart it, type CTRL-Q. (Note that CTRL-S means depress the ‘‘Control’’ key on the keyboard and then hit the *S* key while ‘‘control’’ is still depressed.)

9. You can now edit files, compile programs, or do many other things. The reference manuals given in chapters 8 and 9 of this manual give a brief description of the programs available. However, before rushing off we advise you to adapt the system to your hardware configuration first, as described in the next sections.

10. When you are finished working, and want to log out, type CTRL-D. The

    ```
    login:
    ```

    message will appear, and you or another user can log in again.

11. When you want to shut the computer down, make sure all processes have finished, if need be, by killing them with *kill*. Then type *sync* or just log out. When the disk light goes out, you can turn the computer power off. Never turn the system off without first running *sync* or logging out (which does an implied *sync*). Failure to obey this rule will generally result in a garbled file system and lost data.

3.4. INCREASING THE SIZE OF YOUR RAM DISK

If you have 1M or more of memory, we advise you to increase the size of the RAM disk from 160K to 300K or more. A larger ramdisk allows you to use the RAM disk to copy complete or partial file systems from one diskette to another. It also gives you plenty of space to add a few more utilities to the ROOT file system. Finally, it allows you to compile much larger programs without running out of disk space for the intermediate results. On the other hand it leaves you with less memory to run your MINIX applications. Choosing a RAM disk of 300K leaves you enough memory to recompile most the sources and perform many other tasks.

It is easiest, to use a 360K diskette to carry this enlarged ROOT file system. However, it is a little tricky, to use a 360K diskette to carry a larger ROOT file system. Say you want to make a 512K RAM disk. You may wonder how a 512K RAM disk can be initialized by reading it in from a 360K diskette during the boot procedure. The secret is that the 512K RAM disk is not completely full. Part of it is initially empty so it can be used for scratch files. Only the initialized part (up to 360K) has to be read in. The only problem with this approach is that if you make new root file systems, you should be careful that they do not exceed 360K of data. Failing to do so may damage the file system on your diskette severely.

To install a 512K RAM disk, you must first make a 512K root file system diskette as described below. When MINIX is booted, it looks at the size of the root file system and sets its size accordingly. If you have more than 1 MB, you might even consider making a RAM disk larger than 512K, although only 360K can be initialized at start-up time. To do this, proceed as follows.

1. Take an empty, formatted diskette and label it 10.R512
2. Boot MINIX-ST as described above and login as root. Then type:

   ```
   for i in cpdir mkfs; do cp /usr/bin/$i /bin; done
   /etc/umount /dev/dd0
   ```

3. Insert 10.R512 in drive 0 and type:

   ```
   mkfs –t /dev/fd0 512
   /etc/mount /dev/fd0 /user
   cpdir –msv / /user
   ```

4. Logout by typing CTRL-D.
5. Insert 01.BOOT in drive 0 and type CTRL-ALT-DEL to reboot using 01.BOOT, 10.R512 and 03.USR1.

Do not forget the **–t** option to *mkfs*. It suppresses the check if the new file system fits on the medium. The program *cpdir* will tell you that it skipped the directory */user* to avoid recursion.

By changing the argument *512* to *mkfs* you can adapt the size of the RAM disk. However, if you take a value less than 250 you will run into the problem that *mkfs* allocates not enough inodes to store all the entries of the root file system. If you have 1M of memory and you want to recompile the system a RAM disk of 300K is recommended. Replace the last two occurrences of */dev/fd0* by */dev/dd0* if you prefer to use 720K diskettes, or by */dev/hd3*, or any other hard disk partition, if you want to load the RAM disk from the winchester. Read the section on boot options below if you do.

Note that a copy of the programs *cpdir* and *mkfs* will be present in */bin* on your new ROOT diskette.

3.5. ADAPTING PROGRAMS TO USE EXTRA RAM

As distributed, the C compiler is tuned to work on even the smallest Atari ST configuration. This causes problems if you want to recompile (parts of) MINIX. The first part of the C compiler proper, */usr/lib/cem*, as distributed is configured for a stack size of 40K, but it needs about 70000 bytes more to compile some of the larger source files on the distribution diskettes. It is possible to compile small programs on a 512K machine with the default memory allocation of the compiler.

If you have at least one of the following:

- more than 512K of memory
- two drives, either diskette or hard disk

there are ways to recompile all of MINIX. Note that it is impossible to recompile some parts of MINIX on an ST with only 512K of memory and a single drive.

You are strongly advised to execute the following procedure now if you have more than the minimal 512K of memory.

1. Boot MINIX-ST and login as root.
2. Type:

```
cp  /usr/bin/chmem  /bin
chmem  =35000  /usr/bin/make
/etc/umount  /dev/dd0
```

3. Insert 06.ACK in drive 0 and type:

```
/etc/mount  /dev/dd0  /usr
chmem  =110000  /usr/lib/cem
```

A similar procedure can be executed if you encounter any other program that needs more memory.

3.6. USING SINGLE-SIDED DISKETTES

The distribution contains several 720K diskettes. Most, but not all, Atari ST machines, have a disk drive that can handle 720K diskettes. Only a few older systems can only handle 360K diskettes. If you have one of these systems do not despair. You can split a single 720K diskette into a pair of 360K diskettes on a system with a 720K disk drive. Since you do not have such a system you will have to borrow one from a friend or perhaps your local dealer.

To split 04.USR2 into 13.USR2A and 14.USR2B proceed as follows:

1. Boot MINIX-ST using 01.BOOT, 10.R512 and 03.USR1; login as root.
2. Type:

   ```
   for i in cpdir mkfs rmdir; do cp /usr/bin/$i /bin; done
   /etc/umount /dev/dd0
   ```

3. Insert 04.USR2 in drive 0 and type:

   ```
   /etc/mount /dev/dd0 /user
   mkdir /tmp/a
   ```

4. Now copy files from */user* to */tmp/a*. You should add files to /tmp/a until the command

   ```
   du –s /tmp/a
   ```

   reports a value just below 355.
5. Unmount using:

   ```
   /etc/umount /dev/dd0
   ```

6. Remove 04.USR2 and insert an empty, single-side formatted disk labeled 13.USR2A in drive 0 and type:

   ```
   mkfs /dev/fd0 360
   /etc/mount /dev/fd0 /user
   cpdir –msv /tmp/a /user
   /etc/umount /dev/fd0
   rm –rf /tmp/a
   ```

7. Repeat the same process for the second half of the files on 04.USR2, using an empty, single-side formatted disk labeled 14.USR2B.

Be careful about the subtle difference between */usr* and */user*, between */dev/fd0* and */dev/dd0*, and between *13.USR2A*, *14.USR2B* and *04.USR2*. The result is two 360K diskettes that contain all of 04.USR2. Similarly, you can divide others.

It may happen that you need more than two 360K disks to contain all files of one 720K disk, because the file system itself imposes some overhead that is now doubled. Use three 360K diskettes in those cases.

After you have divided all other 720K diskettes and you have verified your work, you should make another copy of your root diskette (02.ROOT or 10.R512) and modify the file */etc/rc* on that new copy, replacing the line

```
/etc/mount  /dev/dd0  /usr
```

by

```
/etc/mount  /dev/fd0  /usr
```

Now you can use this new 360K version of MINIX just like the original one. However exercise some care when dealing with examples in this chapter or section 7.2, since they assume a 720K version.

3.7. USING A HARD DISK

If you have a hard disk and one or more partitions free for MINIX, you can use it to keep (part of) the distributed diskettes on line. If you have any choice, use a small (512K to 1M) partition 3 (*/dev/hd3*) to hold the ROOT file system that is copied to the RAM disk at boot time. See the section on boot options below. One of the other partitions, for example 4 (*/dev/hd4*), can be as big as 32M and can be mounted on */usr*. It is also possible to keep the root file system on diskette and only use a partition to store the usr file system. In that case you can skip step 6 below. The penalty for keeping the root file system on diskette is an additional disk swap and some additional delay when booting the system. There is no difference in behavior after booting. You could use the whole disk (*/dev/hd5*) (up to 32 MB) as one single MINIX file system, but that would make the disk useless for TOS.

This section describes the steps to set up MINIX on such a system.

3.7.1. Step 1: Backup the Hard Disk

If you are already used your hard disk for TOS, before even *contemplating* installing MINIX, you should make a complete backup of the contents of your hard disk onto diskette or another medium. As a bare minimum, installing MINIX will require erasing one partition of your hard disk, and possibly two. However, to prevent disaster in the event that you make an error during the setup procedure, it is highly desirable that you backup the *entire* disk before you even start. Your files are too valuable to put at risk.

It is worth noting that MINIX has a program, *tos*, that can read TOS diskettes. Thus if you make your backup on diskettes, you will be able to read the files into the MINIX file system after you have completed the hard disk installation.

3.7.2. Step 2: Verify that Your Hard Disk is Atari Compatible

There are a number of different hardware vendors for the Atari ST. Most of their disks work with MINIX. However, some hard disks will not co-operate with MINIX. For example it is known that some of the very first Supra disk controllers will not work with MINIX, due to a bug in the controller. Newer Supra disks (the ones with a SCSI out port) do not have this problem.

To verify that MINIX is indeed able to correctly access your hard disk, boot MINIX as described above, but instead of logging in as *ast*, log in as *root*, using *Geheim* as password (note the upper case *G*). If you are already logged in as *ast*, use CTRL-D to log out, then log in again as *root* (without rebooting). Logging in as *root* makes you the superuser and gives you the sharp sign (#) as prompt instead of the usual dollar sign. The superuser is the system administrator and has special privileges denied ordinary users. To install MINIX on your hard disk, you will need these privileges. Once the installation is complete, you should always log in as *ast*, or create your own login name as described later in this manual.

Once you are successfully logged in as *root*, type:

```
dd if=/dev/hd5 of=/dev/null count=200
```

After a short time, you should get the message:

```
200+0 records in
200+0 records out
```

If you get an error message or no response, MINIX cannot use your hard disk controller.

3.7.3. Step 3: Partition the Hard Disk

Initialize the hard disk (formatting and partitioning) using the tools supplied by Atari, notably the *HDX.PRG* utility. If you have already partitioned your disk before, and you are happy with the partition sizes you can skip this step. Be warned that partitioning the hard disk will destroy all information on that disk. MINIX is not equipped to initialize your disk. The MINIX disk driver requires no special settings of the *pi_flag* and *pi_id* fields (see the Atari hard disk manual), mainly because the Atari hard disk driver code is deficient in properly maintaining the hard disk information found in sector 0. This requires you not to mix up which operating system should operate on which partition, unfortunately. MINIX checks the super block on mounts and it is unlikely that a TOS partition will be accepted. However, writing to a TOS partition by accessing */dev/hd?* directly, although superuser only, is not prevented. Be careful. Similarly, avoid TOS accesses to MINIX partitions. It is a good idea to remove the icons for the MINIX partitions from the TOS desktop.

Another problem is that the *HDX.PRG* seems not to format the last sector on the disk properly, so never use the last sector of the last partition. This is probably a bug

in *HDX.PRG*. So, whenever you make a MINIX file system on the last partition, subtract 1 from the real number of sectors of that partition when calling *mkfs*.

If you have any choice, allocate a small partition 3 of 512K, and a large partition 4 of at least 10M. This setup is assumed in the rest of this section.

3.7.4. Step 4: Make a MINIX File System on Each MINIX Partition

Now that the disk is physically partitioned, it is time to put a MINIX file system on each MINIX partition. To do this, determine the number of sectors in each partition. *HDX.PRG* will have told you the number of sectors when partitioning. The number may not be quite what you had expected due to the use of entire cylinders and rounding effects. Compute the number of 1K blocks in each MINIX partition by dividing the number of sectors by 2 (one block is two 512-byte sectors).

An alternative is to use the command *readall* with the option **–t** on each partition. For example:

```
readall –t /dev/rhd4
```

will tell you the number of 1k blocks on */dev/hd4*. It is possible that during the execution of readall you get a few error messages about unrecoverable disk errors. These error messages can be ignored safely.

To create a file system of, say, 512 blocks of 1K on partition 3 and 10239 blocks of 1K on partition 4, log in as root and type:

```
mkfs  /dev/hd3  512
mkfs  /dev/hd4  10239
```

Notice the 10239 (10240 minus 1) due to the bug in *HDX.PRG* mentioned before. For other MINIX partitions (or sizes) type the analogous commands. Do not run *mkfs* on TOS or other partitions. Be very careful not to make a typing error here, as making a new file system destroys all information on the partition specified.

You can verify that the file systems have been made by typing:

```
df  /dev/hd3
df  /dev/hd4
```

which will report on the i-nodes and blocks present on each file system. The total number of blocks should agree with the number you used in the *mkfs* command.

You can now mount your new file systems. To mount */dev/hd3* (partition 3) on */user*, type:

```
/etc/mount  /dev/hd3  /user
```

To change to */dev/hd3*, type:

```
cd  /user
```

This puts you in the root directory of the partition 3 file system.

3.7.5. Step 5: Check for Bad Blocks

With current manufacturing technology, it is nearly impossible for disk vendors to deliver perfect drives. Almost every drive has some bad blocks on it. If MINIX were to use a bad block in one of your files, you might lose some valuable data, so it is important to locate all the bad blocks before putting any files on the disk and make sure they do not cause trouble.

The scheme used in MINIX is to put all the bad blocks into dummy files, so that the disk space allocator will think they are in use and leave them alone. This method is more efficient than wasting entire tracks as spares, as is sometimes done. Suppose that you have allocated partitions 3 and 4 for MINIX. To locate the bad blocks on partition 3, first log in as *root*, go to the root directory, and unmount the partition, if mounted, by typing:

```
cd /
/etc/umount /dev/hd3
```

It is important that the next commands be executed on the root device, since they will attempt to mount and unmount */dev/hd3*, which will fail if your working directory is there. To locate all the bad blocks, type:

```
readall –b /dev/rhd3 >bad.3
```

Depending on the size and speed of your disk, this operation may take a substantial fraction of an hour. Please be patient. It is possible that during the execution of *readall* you get a few error messages about unrecoverable disk errors. These error messages can be ignored safely. When it is finished, a prompt will appear on the screen. When it does, you can examine the output files using *cat*, *more*, or an editor, for example, by typing:

```
cat bad.3
```

The output will be a shell script that calls *badblocks* with up to seven arguments, each one the number of a bad block. Bad blocks often cluster together. This is normal.

To mark the blocks as bad, type:

```
sh <bad.3
```

When this command finishes, several files full of bad blocks may have been created in the root directory of the device containing the bad blocks. In the example above these files are created in the top level directory of /dev/hd3. After mounting the disk you can examine them by typing:

```
ls –la
```

They will all have names starting with *.Bad_*, followed by some numbers. Do not examine or remove the files. You can now remove the shell script by typing:

```
rm bad.3
```

If you now type:

```
df /dev/hd3
```

you will notice that the number of blocks used has increased by the number of bad blocks found, and the number of free blocks has decreased by the same amount.

If you have more MINIX partitions, go to the root directory and unmount the current partition. Then mount the next partition and repeat the same process. If the next partition is 4, the sequence is as follows (where the text starting at the # signs are just comments):

```
cd /                          # go to the root directory
/etc/umount /dev/hd3          # if still mounted
readall –b /dev/rdh4 >bad.4# find the bad blocks on partition 4
sh <bad.4                     # mark the bad blocks on partition 4
rm bad.4                      # remove the shell script
/etc/umount /dev/hd4
```

There is a small chance that a bad block will occur in the i-node list of a new file system. If this occurs, you must go back to Step 3 and repartition the disk with different sizes, trying until all of the i-node blocks are good.

3.7.6. Step 6: Initialize the Root File System

When MINIX boots, it needs a root file system. By default, this root file system is read from a 360K diskette and copied into memory as a RAM disk. If you have a hard disk an easier alternative is to read the root file system from a hard disk partition, preferably */dev/hd3*, and copy it into the RAM disk.

This requires you to make a copy of the root file system onto */dev/hd3*. In the discussion below we will put the root file system on the 512K partition */dev/hd3* on which we have already made an empty file system above. However, you could equally well use another partition, but take care that the size of the file system you make on that partition (the argument to *mkfs*) is used as the size of your RAM disk.

The procedure below is actually rather similar to the procedure described before to increase the size of your RAM disk. Proceed as follows:

1. Boot MINIX-ST with 01.BOOT, any ROOT (02.ROOT or 10.R512) and 03.USR1 and login as root. Then type:

```
for i in cpdir mkfs chmod; do cp /usr/bin/$i /bin; done
/etc/umount /dev/dd0
/etc/mount /dev/hd3 /user
cpdir –msv / /user
```

2. Logout by typing CTRL-D.

You can now test if the new root file system really can be used to boot from. Insert 01.BOOT in drive 0 and type CTRL-ALT-DEL to reboot. You will be confronted again with the message:

```
Insert ROOT diskette and hit RETURN (or specify bootdev)
```

As alternative for the insertion of 02.ROOT or 10.R512 as second step in the boot procedure you now have three option:

1. Keep the 01.BOOT diskette in drive 0, and hit RETURN. MINIX-ST will not find a file system on the diskette and will try to load the root file system from hard disk partion 3, precisely where we have created our new root file system.
2. Reply with

   ```
   3,3
   ```

   to override the default by loading the root file system from hard disk partition 3.
3. Reply with any other drive specification, like

   ```
   3,2
   ```

   if you want to load the root file system from partition 2, for instance.

3.7.7. Step 7: Initialize */usr*

The next step is creating all the directories. A shell script called */etc/setup_usr* has been provided to do most of the work. It creates a large number of directories. Next, it copies files from the distribution diskettes to the */usr* tree on the hard disk. It asks for 03.USR1 to 09.SRC3 in sequence. Just follow the instructions that appear on the screen until the ‘‘Installation completed’’ message appears. To perform the installation be sure you are logged in as *root*. We assume that you have setup

/dev/hd4 as described above, and that */dev/hd4* contains at least 10M. Then, proceed as follows:

1. Boot MINIX-ST using 01.BOOT, any ROOT (02.ROOT, 10.R512 or hd3) and 03.USR1 and login as root.
2. Type the commands:

```
for i in cpdir test echo; do cp /usr/bin/$i /bin; done
/etc/umount /dev/dd0
/etc/mount /dev/hd4 /usr
/etc/setup_usr
```

3. Follow the instructions displayed by the *setup_usr* script. If your partition is smaller than 10M, the best thing to do is to install only the binaries onto the hard disk. Type *quit* when the system asks you to insert disk 07 (07.SRC1). Installing only the binaries will require 4M.

Except for the boot diskette and the tos diskette, all the distribution diskettes are normal MINIX file systems that you can mount and inspect if something should go wrong. When this shell script finishes, the entire MINIX file system will be installed on the hard disk. Most of the files on the distribution diskettes are compressed files (with suffix *.Z*) or compressed archives (with suffix *.a.Z*). If, for some reason, installation fails part way through, you may be left with some *.a.Z*, *.a* or *.Z* files on the disk A file *file.a.Z* can be decompressed using :

```
compress –d file.a.Z
```

If the result is an archive (with suffix *.a*), you can extract the files from the archive with the *ar* command, for example:

```
ar x file.a
```

At this point the files *file.a.Z* and *file.a* can be removed. The only archive that you *must* keep as an archive is *libc.a* as the C compiler expects it this way. Do not extract the individual files from it!

From now on you can mount */dev/hd4* at boot time as */usr* by making a small change in */etc/rc* found on the ROOT file system (diskette or winchester). Use *mined* (see chapter 9 on how to use *mined*) to change the first two lines that read:

```
/bin/getlf ″Please insert /usr diskette in drive 0. Then hit RETURN.″
/etc/mount /dev/dd0 /usr
```

by a single line that reads:

```
/etc/mount /dev/hd4 /usr
```

Inserting diskette 03.USR1 will no longer be necessary at boot time.

3.8. USING A MEGA ST

The Mega ST series is internally quite similar to the Atari 520 ST and 1040 ST machines. It has more memory, which is automatically supported by MINIX-ST. It has a blitter chip, but currently MINIX-ST does not support it. Another standard feature is the battery powered real time clock. To eliminate the need to type the date each time the system is boot, a small program that reads out the current date and time from the real time clock, and sets the MINIX time accordingly has been provided. If you have a Mega ST you are advised to adapt the file */etc/rc* so that it will use that program *megartc* whenever you boot. Replace the line that reads:

```
/usr/bin/date –q </dev/tty
```

by the following two lines:

```
/usr/bin/megartc
/usr/bin/date
```

Note that *megartc* is found on 03.USR1. This change has the following effect. The program *date* queries the terminal for the date and then installs the date. The program *megartc* takes the date from the real time clock instead of asking for it from the terminal. The second line causes the date to be printed.

As an aside, please note that any other commands inserted in the file */etc/rc* will be executed before the system is booted. However, when inserting commands there, be sure that they do not require programs or files that are on diskettes that have not yet been mounted.

3.9. USING A DISK CONTROLLER BASED CLOCK

Since the original Atari ST did not contain a battery powered real time clock, quite a number of add-on clocks have appeared on the market. MINIX-ST supports the real time clock from Weide. It also supports the clocks available on various third party disk controller boards, but only if you recompile your kernel with the **–DCLOCKS** option in the kernel Makefile turned on. See chapter 7 for an explanation of rebuilding the kernel. For both types of clocks a small program that reads out the current date and time from the real time clock, and sets the MINIX time accordingly has been provided. In both cases you are advised to adapt the file */etc/rc* so that it will read the real time clock whenever you boot. If you have a Weide real time clock replace the line that reads:

```
/usr/bin/date –q </dev/tty
```

by the following two lines:

```
/usr/bin/weidertc
/usr/bin/date
```

If you have a disk controller with a real time clock and have a modified operating system (*MINIX.IMG*) on your 01.BOOT diskette, replace the same line by:

```
/usr/bin/diskrtc controller
/usr/bin/date
```

where *controller* is one of *supra*, *icd*, *bms1* (for a BMS 100 controller) or *bms2* (for a BMS 200 controller). Note that also these programs are found on 03.USR1.

3.10. BOOT PROCEDURE OPTIONS

The boot sequence we have described so far always starts with a 360K BOOT diskette in drive 0, followed by a 360K ROOT diskette in drive 0. Between the BOOT and ROOT diskette you have always answered the question:

```
Insert ROOT diskette and hit RETURN (or specify bootdev)
```

by hitting RETURN. If the ROOT file system is found on another device you may specify that device as:

major,minor

where *major* is a decimal number specifying the device type and *minor* is a decimal number specifying the drive and/or partition. These *major,minor* pairs correspond with the numbers you see in the output of:

```
ls –l /dev
```

Some of the useful combinations are:

Major	Minor	Device	Description
2	0	fd0	360K diskette in drive 0
2	1	fd1	360K diskette in drive 1
2	8	dd0	720K diskette in drive 0
2	9	dd1	720K diskette in drive 1
3	1	hd1	partition 1 of hard disk 0
3	2	hd2	partition 2 of hard disk 0
3	3	hd3	partition 3 of hard disk 0
3	4	hd4	partition 4 of hard disk 0
3	5	hd5	complete hard disk 0

So, if ROOT is found on a 720K diskette in drive 1 the second line of your screen will look like:

```
Insert ROOT diskette and hit RETURN (or specify bootdev) 2,9
```

If you specify nothing or anything illegal, MINIX will check two default devices in sequence. First it tries to read the super block of the ROOT file system on 2,0 (360K diskette in drive 0). Only if that fails (read error or illegal super block) it tries 3,3 (partition 3 of hard disk 0). That is why we advised you to use */dev/hd3* as copy of the RAM disk.

One of the more exotic options of the boot sequence is to read the MINIX operating system itself from a TOS file, not using the BOOT diskette. On the diskette 00.TOS you find a TOS program *MINIX.PRG* that takes as first argument the name of a TOS file, default *MINIX.IMG*, that contains the operating system. You can create the file *MINIX.IMG* yourself by reading enough sectors from the BOOT diskette, starting with sector 0, but it requires at least one other diskette, hard or RAM disk besides a:. The procedure below assumes that you have a TOS RAM disk named *m:*. Proceed as follows:

1. Start TOS.
2. Insert a copy of 00.TOS, in drive 0.
3. Double click icon FLOPPY DISK A.
4. Double click *COMMAND.TOS* on A:

   ```
   rflop a: m:\minix.img 100000
   ```

5. Insert protected 01.BOOT if you are asked and hit RETURN.
6. When done, put *MINIX.PRG* and *MINIX.IMG* onto a TOS diskette

The third argument to *RFLOP* is the number of bytes to read. 100000 is more than sufficient for the operating system as distributed. You can now copy *MINIX.PRG* and *MINIX.IMG* to a TOS partition of the hard disk. Assuming that you normally boot TOS from the hard disk, you can subsequently switch to MINIX by double clicking *MINIX.PRG*. If you want to switch back to TOS you logout by typing CTRL-D. If you see the prompt *login:* again, type CTRL-ALT-DEL.

3.11. UNPACKING THE SOURCES

The sources, except the compiler and *elle*, are on the SRC diskettes. These diskettes are normal MINIX file systems, which you can mount using the command:

```
mount /dev/dd0 /user
```

The files on the distribution diskettes are compressed archives (with suffix *.a.Z*). If you want to extract the sources from a file *file.a.Z* you should first copy this file to either an empty diskette, or to the RAM disk, if the latter is large enough. Typically about 4 times the size of the compressed file is required when extracting the sources. If later on you want to recompile the sources even more space may be required. That is why you first should copy the compressed source file to an empty diskette. Your copy of *file.a.Z* can be decompressed using:

```
compress –d file.a.Z
```

After decompressing you can remove your copy of *file.a.Z*. Now you can extract the files from the archive with the *ar* command, for example:

```
ar x file.a
```

At this point all files from the archive are extracted. You can now remove *file.a* since it is no longer needed.

3.12. THE TOS TOOLS

Several tools have been developed for TOS. In the early stages of the MINIX-ST port TOS was used as the development environment. That forced us to port tools like *mkfs* and *build*, and to develop the programs *minix* and *relmix*. Later, when the native MINIX-ST C compiler became available, we could use MINIX-ST itself for further development. Rather than simply discarding the TOS tools, we have included them in the distribution for the benefit of people wishing to do further MINIX-ST developments using TOS. Below we describe these tools in the same style as the MINIX commands.

Command: **BUILD.PRG – build *MINIX.IMG*** out of its constituent parts
Syntax: **build** *bootblok kernel mm fs init menu minix.img*
Flags: (none)

Build takes the six constituent parts and produces the MINIX-ST operating system image. That image, if written onto a diskette starting at sector 0, is bootable on the Atari ST. Alternatively, the program *MINIX.PRG* can be used once TOS is up and running.

Command: FIXKEYS.PRG – patch BOOT diskette for TOS keyboard table
Syntax: **fixkeys** [**–d**] [**–o**] *drive*
Flags: **–d** Double-sided diskette
–o Accept not only a: and b:
Example: fixkeys a: # Modify BOOT diskette in drive a:

Fixkeys patches the keyboard tables of the currently active version of TOS into the MINIX-ST operating system image as normally found on the BOOT diskettes. It can only operate on diskettes, not on file images.

Command: KEYTBL.TTP – display the keyboard tables
Syntax: **keytbl.ttp** [*file*]
Flags: (none)

Keytbl writes the keyboard tables to the file whose name is gives as a parameter (or to standard output if no parameter is present). This file can be used when recompiling the kernel. Refer to chapter 7 for details on how to recompile the kernel.

Command: MINIX.PRG – boot MINIX-ST from an image on file
Syntax: **minix** [*image*]
Flags: **(none)**
Example: minix minix.img # boot MINIX-ST from minix.img

Minix allows you to boot MINIX-ST if TOS is already up and running. It reads the operating system image from a TOS file into memory, copies the image to address 0 and jumps to the address found at location 4. There is no way back to TOS, except by rebooting the machine.

Command: MKFS.PRG – make a MINIX-ST file system
Syntax: **mkfs** [**–dol**] *drive prototype*
Flags: **–d** Double-sided diskette
–o Overwrite: accept not only a: and b:
–l Make a listing on standard output
Examples: mkfs a: proto # Make a file system on drive a:
mkfs -d b: 360 # Make empty 360 block file system

Mkfs builds a file system and copies specified files to it. See chapter 8 for a description of the proto file syntax. The files used to initialize the new file system should conform to the TOS syntax, including backslashes and drive specifications.

Command: RELMIX.PRG – change loadfile from .68K to .MIX format
Syntax: **relmix** [+*amount*] [–*amount*] [=*amount*] *prog.68k prog.mix*
Flags: **+** Increase memory allocation
– Decrease memory allocation
= Set memory allocation
Example: relmix =2000 x.68k x.mix # Make MINIX-ST style loadfile

The Alcyon 4.14 C compiler, part of the Atari ST developers kit, produces a loadfile in .68K format. A simple transformation of the header, removal of the symbol table, and a transformation of the relocation information as performed by the RELMOD.PRG program (also part of the developers kit) does the trick.

Command: RFLOP.PRG – read bytes from diskette
Syntax: **rflop [–d][–o]** *drive file bytes*
Flags: **–d** Double-sided diskette
–o Accept not only a: and b:
Example: rflop a: minix.img 100000 # Read *minix.img* from BOOT diskette

An arbitrary number of bytes is read from the diskette and written to a TOS file. Reading always starts at sector 0.

Command: WFLOP.PRG – write bytes to diskette
Syntax: **wflop** [**–d**] [**–o**] *drive file*
Flags: **–d** Double-sided diskette
–o Accept not only a: and b:
Examples: wflop a: minix.img # Make BOOT diskette

A TOS file is written to a diskette, starting at sector 0. This overwrites the information in sector 0 used by TOS to determine the type of the diskette.

3.13. TROUBLESHOOTING

As a user of MINIX-ST you may be confronted with some of the error messages the system can produce. The following subsections give guidelines on you how to react. It also explains how you can use the built-in debugging aids.

If you have problems booting the system, try the following steps: power down the machine, wait 10 seconds, insert the BOOT diskette in drive 0, and power up the machine. If you have a hard disk and normally boot from the hard disk directly, you may either force booting from the diskette as described in the Atari hard disk manual, or start MINIX-ST using the supplied TOS program *MINIX.PRG*, as described in section 3.9 of this manual.

Either way, the screen should turn black and you will see two lines printed on the top of the screen, asking you to insert the ROOT diskette. If these lines do not appear, the BOOT diskette is probably damaged.

Hitting RETURN at this point should give one more line. If not, you might suspect the keyboard or the BOOT diskette. Normally, when the root file system is being read in, regular progress reports appear on the screen. If not, the diskette drive may not be working correctly with the MINIX-ST diskette driver (e.g., because your diskette controller does not generate interrupts as it should). This should not be a problem with all known Atari ST production models, but we have heard about some problems with very old development machines. If disk error messages appear on the screen, your drive may need slower step rates than usual. Official Atari diskette drives should work correctly.

If the system behaves funny or even crashes while loading the root file system, the ROOT diskette is suspect. It might be corrupted or too big for this machine. If so, try it with (a copy of) your 02.ROOT diskette.

If you have gone through these critical initial steps you should not have any problems getting MINIX-ST booted, since the essential resources, the diskette, the keyboard and the screen are probably all right.

3.13.1. Error Messages

Many of the error messages are also found in MINIX-PC. Here we list the MINIX-ST specific ones. In all cases % followed by a letter gives the *printf* format of the number.

Three messages are printed by the kernel if commands running on top of the operating system itself encounter problems, such as unsolicited hardware traps and stack overflow. These three are:

- **sig=%d to pid=%d at pc=%X**
  Generated if bus errors, segmentation faults, illegal instructions or funny traps are encountered,

- **Stack low (pid=%d,pc=%X,sp=%X,end=%X)**
  If a stack overflow has happened or is about to happen, and

- **Unexpected trap. Vector = %d**
  This may be due to accidentally including
  a non-MINIX library routine that is trying to make a system call.
  If any of the trap instructions is executed that is not used by MINIX-ST. In both cases, the system will continue, but the program is likely to be aborted with a core dump generated on the file *core*.

A number of messages announce unexpected hardware events, sometimes only a warning, sometimes more serious, but not immediately fatal. In this category fall:

- **fd%d: timeout**
  No diskette in drive (you have 15 seconds to insert one)

- **fd%d: read: dma status = 0x%x**
  DMA error on diskette read request

- **fd%d: read sector %d: fdc status = 0x%x**
  Diskette controller error on read request

- **fd%d: write protected**
  Writing to write-protected diskette

- **fd%d: write sector %d: fdc status = 0x%x**
  Diskette controller error on write request

- **fd%d: recalibrate failed. status = 0x%x**
  Cannot find track 0

- **hd: read: drive=%d sector=%D status=0x%x**
  Hard disk error on read request

- **hd: write: drive=%d sector=%D status=0x%x**
  Hard disk error on write request

- **DMA interrupt discarded**
  Unsolicited interrupt from device on DMA bus

- **midi interrupt: status=%x, data=%x**
  Unsolicited interrupt from MIDI interface

- **Fake interrupt handler for %s. trap = %02x**
  Unsolicited interrupt from:
  timint,00: timer A of MFP chip
  timint,01: timer B of MFP chip
  timint,03: timer D of MFP chip
  siaint,00: MFP RS232: char received
  siaint,01: MFP RS232: receive error
  siaint,02: MFP RS232: char transmitted
  siaint,03: MFP RS232: transmit error
  iob,01: MFP RS232 Data Carrier Detect

iob,02: MFP RS232 Clear To Send
iob,03: unused
iob,06: MFP RS232 Ring Indicator
iob,07: Monochrome Monitor Detect

- **Printer is not available**
  Ready bit off: not connected or off line

- **printer: still busy**
  Interrupt received, but not ready

More serious conditions cause a system panic. A message is printed and an infinite loop is entered. Only a reset helps: push the RESET button (sometimes CTRL-ALT-DEL works as well). The most important MINIX-ST specific kernel panics are:

- **dma:ASSERT(%s) failed**
  Consistency checking in stdma.c

- **fd:ASSERT(%s) failed**
  Consistency checking in stfloppy.c

- **Nonexisting interrupt. Vector = %d**
  A trap via one of the vectors that is unassigned

- **Unexpected interrupt. Vector = %d**
  A trap via one of the autovectors not used by the ST

- **trap via vector %d**
  A synchronous trap in kernel mode

- **no shadow?**
  Two processes share an ORIGINAL, but neither points to a SHADOW

- **rmshadow: cannot handle physio shadows**
  SHADOW with p_physio set must be copied to ORIGINAL.

- **only shadow(s)**
  All that share ORIGINAL have SHADOW set

- **tty_init: unknown terminal %d**
  For all NR_TTYS an initialization routine must be called

The important file system panics are:

- **Invalid root file system**
- **RAM disk is too big. # blocks = %d**
- **Root file system corrupted. Possibly wrong diskette.**
- **init: can't load root bit maps**
- **Disk error loading BOOT disk %d**

For all these errors retry booting with (a copy of) 02.ROOT.

3.13.2. Debugging Aids

Some of the internal tables can be inspected by special key combinations. The keyboard driver recognizes the following key combinations and calls debugging routines in the kernel:

CTRL-ALT-F1	dump of the process table
CTRL-ALT-F2	dump of the memory map
CTRL-ALT-F3	dump of the status of the current process

The process table shows the current and lowest stack pointer detected, the CPU time spend in user mode and system mode, and the memory slot occupied for each process, including kernel tasks, as well as other information.

The memory map shows (for user processes only) the location and length of the text, data and stack segments, and the shadowing fields *p_shadow*, *p_nflips* and *p_physio*.

The status of the current process shows the register values most recently saved, a memory dump around the location of the program counter, and a memory dump around the location of the stack pointer. The memory dumps can be used to give a stack trace and, by manual disassembly, the instructions executed most recently.

The CTRL-ALT-F6 key combination toggles an option to dump the same tables whenever the message

sig=%d to pid=%d at pc=%X

is printed and whenever the system panics. By default the ''automatic table dump'' option is OFF.

The CTRL-ALT-F5 key combination toggles an option to send all kernel generated output not only to the screen but also to the line printer. By default the ''kernel output to printer'' option is OFF. If the printer is offline, the printing is temporarily suppressed. If the ''kernel output to printer'' option is ON, and the printer switches from online to offline, it may take a few seconds to detect this, since initially it looks similar to a printer buffer full condition. Be patient.

The CTRL-ALT-F4 key combination toggles an option to send all kernel generated output to the screen. By default the ‘‘kernel output to screen’’ option is ON.

A good procedure, if you encounter a problem and you want to spot it, is to isolate the problem such that it is reproducible. Then, insert paper in the printer and toggle CTRL-ALT-F5 and CTRL-ALT-F6 to capture the debugging information on paper while you reproduce the error situation.

If the problem is in a command the *core* file contains a memory dump at the time of the crash. These post mortem dumps can be analyzed using the mdb debugger. Refer to chapter 9 for a description of *mdb*.

If problems are encountered in the MINIX-ST driver, you have a chance that that driver has debugging statements coded in. By changing either the #DEBUG or #TRACE definitions, you can effectuate these statements, but only after recompilation. Refer to chapter 7 on how to recompile MINIX