NAME

dosminix, mkfile − Running Minix under DOS

SYNOPSIS

C:\MINIX> **boot disk0.mnx** (Typical example)
C:\MINIX> **mkfile** *size disk*

DESCRIPTION

This text describes running Minix under DOS. The DOS version of the Boot Monitor, described in **monitor**(8), grabs as much memory as DOS is willing to give, loads Minix into that memory from the active partition of a "file as disk", and jumps to the Minix kernel to let Minix take control. As far as DOS is concerned Minix is just a part of the **boot.com** program.

In the example above **disk0.mnx** is the "file as disk". It is a file of many megabytes that is used by Minix as a disk of four partitions. These partitions will normally be **/dev/dosd1** through **/dev/dosd4**, with **/dev/dosd0** for the whole "disk". The Boot Monitor will set the **dosd0** boot variable to the name of the disk (its first argument), the root file system will be the active partition, usually **dosd1**. It is better to use the special name **bootdev** to indicate this device, usually in the setting **rootdev**=**bootdev**.

Once Minix is running it will operate the same as if started from a regular disk partition until it is shut down. On shutdown from protected mode it will return to the Boot Monitor prompt, and with the **exit** command you leave the Boot Monitor and return to DOS. Shutting down from real mode will reboot the machine, just like when run from a disk partition. (This more or less crashes DOS, but DOS is used to such abuse.)

EMM386

Minix can't run in protected mode (286 or 386 mode) if DOS is using a memory manager like **EMM386**. You can either temporarily comment out EMM386 from **CONFIG.SYS**, or you can press **F8** on startup to bypass CONFIG.SYS. This is only possible with the later DOS versions.

Windows 95

Press F8 at startup to make the boot menu visible. Choose "**Command prompt**", or "**Safe mode command prompt**" to run DOS. Use the "safe mode" if EMM386 is started in CONFIG.SYS.

Typing F8 at the right moment isn't easy, so you may want to change the way Windows boots by editing the **MSDOS.SYS** file found in the root directory of your Windows system. This is alas not trivial. Open a window on your main drive, click on "**View**" and choose "**Options**." In the Options window choose "**View**" and enable "**Show all files**". The MSDOS.SYS file should now be visible, among several other hidden files. Right-click on the MSDOS.SYS icon, choose "**Properties**" and disable "**Read-only**". Bring MSDOS.SYS into a simple text editor such as Notepad. In the **[Options]** segment add the following lines (or change existing lines into):

```
BootMenu=2
BootMenuDelay=5
```

The first setting makes the Windows boot menu always visible, and the second line changes the delay before booting to 5 seconds. Take care not to change anything else, or things will go horribly wrong. Save MSDOS.SYS and exit. Don't forget to make MSDOS.SYS read-only again, and also hide all the hidden files again, unless you like it this way.

DOS compatibility box

The 16-bit version of standard Minix can be run in real mode in a DOS box. This is somewhat surprising, because it means Windows 95 simulates devices like the keyboard, timer, and interrupt controller well enough to fool Minix into thinking that all is well. Alas it doesn't work as well under Windows NT. Keypresses get lost if you type to fast, and using the floppy occasionally locks Minix up. This is a bit disappointing, because it is the only way to run Minix under NT. Under Windows 95 one is better off putting the system in DOS at boot and then to run Minix in protected mode.

One thing that is better under NT is that the Boot Monitor is able to get a so-called "Upper Memory Block", thereby raising useful memory to about 750K. Windows 95 however hogs leftover UMB memory in a process named **vmm32**, whatever that may be. To get some of this memory you can put **BOOT /U** at the start of **autoexec.bat**. The monitor will grab a 64K UMB if it can get it, and keep that memory safe for use by Minix when it is later started from Windows.

The easiest way to start Minix is to give all Minix disk files the suffix **MNX**. Doubleclick on the disk you want to run to make the "**Open With**" window appear. Click on "**Other**" and browse to the **BOOT.COM** program. Set the name of the .mnx files to "**Minix "disk" file**" in the description box if you want everything right. In the future you can just click on a Minix disk file to run it, you don't have to start a DOS box first. (To make it perfect use "View", "Options", "File Types", choose "Minix "disk" file", "Edit", "Change Icon", "Browse", select MINIX.ICO.)

When Minix shuts down it will try to reboot what it thinks is a PC. Windows seems to assume that the DOS session has exited. Right-click on the BOOT.COM program, "Properties", "Program", and enable "Close on exit" to make the DOS box disappear automatically when Minix thinks it reboots. You may also want to lock the font to **7x12**, or any other font that isn't ugly.

Minix disk files are opened in a write-exclusive mode. A second Minix session can only open it read-only, which may lead to a "can't open root device" error.

Mkfile

Minix disk files can be created or resized with the **mkfile** utility. Its two arguments are the size and name of the disk file. The size is a number optionally followed by the letter **k**, **m** or **g** to specify kilobytes, megabytes, or even gigabytes. So the call

mkfile 50m disk5.mnx

will create a 50 megabyte file named **disk5.mnx**. If the file already exist then it is shrunk or grown to 50 megabytes. No data is lost if the file is grown. If the file is shrunk then only the data that is cut off is lost. These features allow one to inrease the size of a Minix /usr partition with the following recipe:

copy disk0.mnx disk0.new	Copy the disk to disk0.new
mkfile 100M disk0.new	Enlarge to 100 megabytes
boot disk0.mnx	Boot the old "disk"
[ESC]	Get the attention of the monitor
dosd5=disk0.new	/dev/dosd5 becomes disk0.new
boot	
...	
login: root	
part	Choose dosd5, move to the Size field of dosd7 partition, hit 'm' to fill it out to the end of the "disk". Write and quit.
mkfs /dev/dosd7	Recreate the file system, but larger
mount /dev/dosd7 /mnt	
cpdir -v /usr /mnt	Copy /usr to the new disk's /usr to be
shutdown	Back to the monitor
exit	Back to DOS
ren disk0.mnx disk0.old	
ren disk0.new disk0.mnx	Replace old by new
boot disk0.mnx	Run the larger system

Now Minix runs from a larger "disk". Don't worry if it claims to have crashed, there wasn't a "shutdown" entry in /usr/adm/wtmp at the time it was copied.

The above recipe is for a ordinary standard Minix installation with /usr on the second and last partition.

Backups

In the recipe above you saw how simple it is to create a new system, just copy a disk file. It is equally simple to make a backup, you just copy the disk file. To make a test system: copy the disk file. To make another test system: copy the disk file. Let friends have their own Minix: copy the disk file again. (Exciting, eh?)

You may want to save a Minix disk file in a ZIP file to save space. It may look as a good idea to first run **make clean** in **/usr/src** to remove all the binary junk, but alas that has no effect at all. The disk file is compressed under DOS, and there it is unknown which blocks are in use and which are free. With the following trick you can make those deleted blocks compress really well:

```
cd /usr/tmp
echo >junk
while cat junk >>junk; do :; done
sync
rm junk
```

After these commands all free blocks contain newlines. Long runs of the same byte happen to compress by a factor 1000, so the unused disk blocks will almost disappear in the ZIP file.

FAT driver

The dos disk driver, described in **dosd**(4), has two identities. By default you get the "**file**" driver, that uses DOS file I/O calls to access a large DOS file as a disk. The other alternative is the "**FAT**" driver. The FAT driver sits on top of an ordinary Minix disk driver, and interprets a partition as a FAT (File Access Table) file system to find a file to use as a Minix disk. The result has the same effect as the file driver, except that no costly calls to DOS are made. To enable this feature you have to use the following Boot environment settings:

```
dosd = fat
dosd0 = hd1:\minix\disk0.mnx
```

The **dosd** setting tells Minix to use the FAT driver, and the **dosd0** setting tells the Minix device and DOS file name to use. Disk I/O should be sped up nicely by this change, although typical use of Minix doesn't require fast disk I/O, so the difference won't be too noticable.

Support for FAT-32 (big file system support added in the later Windows 95 releases) has not been tested very well. The FAT-12 and FAT-16 code has been used a lot, and seems safe. Note the risks inherent in these drivers: The file driver uses simple DOS file I/O calls, leaving it to DOS to know its own file system. The FAT driver interprets FAT file system structures by itself. Minix booted from a real hard disk partition can only use DOS disk files through the FAT driver.

SEE ALSO

dosd(4), **monitor**(8), **usage**(8).

NOTES

Use at your own risk.

BUGS

Hasn't been tried under Windows 98 yet.

Pray the deity of your choice will forgive you for running a UNIX-like system as an ordinary DOS program. The author of this code is already doomed. When his time comes the daemons wi*(&%*$%*&
Memory fault − core dumped

AUTHOR

Kees J. Bot (kjb@cs.vu.nl)