miniBSD - reducing FreeBSD **(last update: 08/25/2005)**

Portuguese version - translation by Gustavo Fukao
Russian version - translation by Ilia Kuliev

Note: this guide refers only to FreeBSD 4.x, which is now officially obsolete. David Courtney has created updated versions for FreeBSD 5.x and 6.x:

- **miniBSD 5.x Guide**
- **miniBSD 6.x Guide**

For a long time, I knew one day I would get myself a little SBC (Single Board Computer) - just to try it out for fun. When I saw those that Soekris Engineering offers, I knew that their net4501 SBC was exactly what I wanted. A low power CPU, three 10/100 NICs and a Compact Flash card instead of a hard disk - the perfect hardware to build my own firewall/router.

So I ordered one of those. While waiting for it to arrive, I wondered what kind of operating system I could run on it. I knew it had to be some UNIX flavour, preferably FreeBSD. I wanted the board to act as a router with NAT and firewall, as a small HTTP and FTP server and as a DHCP server for my LAN. I did a "minimal" install of FreeBSD on a normal PC, but found that it requires 80 MB. My goal was to fit everything on a 64 MB CF card and still have ample space for user data. I know there are some Linux distributions that would fit my needs, but I don't like Linux much anymore. There is a project, PicoBSD, which aims at fitting FreeBSD on a single floppy disk, but that's too minimalistic for me - I wanted something in between a FreeBSD minimal installation and PicoBSD.

I did another minimal FreeBSD install and started looking for things I could remove in order to save space. After much tinkering, my "miniBSD" only weighed 22 MB (all binaries linked dynamically) and still had all the functionality I wanted (including ssh, FTP, perl and all the basic commands one expects on a reasonable UNIX system). Without perl, it fits in about 12 MB.

So here's a guide on how to start if you're looking for the same thing. I assume that you already have enough knowledge about FreeBSD to decide on your own, for example, which device files or binaries you need. **Note also that this is not a complete guide**, and once you're done with it you'll still need to customize your miniBSD to render it useful. I assume you're trying to achieve the same goal - installing FreeBSD on a net4501 - but with a few changes this guide also applies to other installations (e.g. on CD-ROM, hard disk, ...).

I now have my net4501 up and running as a PPPoE capable gateway with NAT and Firewall, as well as a web server (thttpd) and FTP server. It works rock-solid!

If you have any suggestions, success reports or whatever, please let me know at <mk@neon1.net>. Please be aware that I can't answer all newbie's questions - search at http://groups.google.com/ - I can almost always find the answer to my questions there.

Finally, I'm sorry for any spelling/grammar mistakes - English isn't my native language (if you want to proof-read this document - go ahead! :).

Table of contents

Revision history

8/25/2005

- removed FreeBSD 5.x-specific information from the guide (see section 1 for details)

8/21/2005

- added a hint contributed by David Courtney to section 1.1

6/05/2005

- added a hint contributed by Mars G. Miro to section 7
- posted a message from Jeremy Cooper (section 11) that explains possible problems with C/H/S mismatches

5/19/2005

- added a hint contributed by Anders Trobäck to section 8

5/19/2005

- added a hint contributed by Anders Trobäck to section 8

5/08/2005

- added some hints about using bsdlabel (contributed by Steve Lin)

3/28/2005

- added a hint contributed by James Bowman to section 7

2/09/2004

- Gregory Harris sent an improved mkmini.pl which automatically makes directories that do not already exist, and a fixed minibsd5.files that should work with FreeBSD 5.2. Thanks, Greg!

8/27/2003

- added a hint contributed by Tom Thompson about pkg_info to appendix A

8/01/2003

- mentioned the lack of the command "disklabel" in FreeBSD 5.1 (now called "bsdlabel" and has different flags) in section 1 (pointed out by Kent Berggren)

7/06/2003

- added a hint contributed by Stuart Henderson concerning the use of a stripped-down termcap file to section 5
- added a note concerning FreeBSD 5.1 to section 8

3/04/2003

- removed appendix D again and added the information relating to FreeBSD 5.0 to the corresponding sections
  (based on my own experiences)

2/28/2003

- added appendix D based on information contributed by Ian Gorrie

2/8/2003

- section 1.1: added a hint contributed by Murray Taylor

11/16/2002

- section 3: added a note about the fact that the developer distribution is required for recompiling the boot loader, and also uploaded a precompiled /boot/loader for 4.7-RELEASE
- section 6: added information on the serial console speed settings

10/26/2002

- added -o noatime to mount command in section 13

10/11/2002

- added new kernel options for FreeBSD 4.7

10/04/2002

- added a hint sent in by Roddy Collins that /etc/resolv.conf has to be symlinked to /tmp/resolv.conf if you're using dhclient.

09/22/2002

- removed /etc/rc.syscons from the /etc file list - it's not needed and can actually cause errors because it may use vidcontrol, which of course is pointless since there's no video card in the net4501.
- added a note about the warnings displayed by mount_mfs to section 13

07/25/2002

- a painful experience made me add a note to section 4 that if you go and "make install" with NOSHARED=no set in a real FreeBSD system where /usr sits in a separate partition, you'll be in for a nice surprise on the next reboot - the kernel will find /sbin/init to be linked dynamically with no libraries to be found anywhere (since /usr is not mounted at that time, yet)
- changed ssh-keygen command lines in section 13 to reflect the new syntax (and the additional key) in FreeBSD 4.6.1

07/10/2002

- added a hint on how to enable getty on the serial line to section 8 (so you can actually login via the serial console)

07/07/2002

- added chapter 1.1 (chroot jail) - thanks to Jerry Hicks for the hint
- totally rewrote section 7 (Copying the libraries), together with a new script to compile a list of required libraries by running ldd on all binaries (inspired by a shell script that a reader sent me)
- inserted section 4 (Dynamic vs. static executables) - compiling the binaries in /bin and /sbin (and some binaries in /usr/bin) saves a few MBs (suggested by Eugene Grosbein)
- added a warning about mount -u -o ro /

07/06/2002

- changed relative to absolute paths in some commands
- added information about swap partitions

---

1. Installing FreeBSD

First of all, you need a standard FreeBSD installation on a normal PC or laptop. I will not cover this topic in detail, because I assume you know FreeBSD to some degree (or are willing to learn :). A minimal installation with kernel sources will suffice. This guide is based on FreeBSD 4.8, although it should work with earlier and newer 4.x versions, too.

If you're looking for a FreeBSD 5.x-specific version of this guide, try David Courtney's version. The 5.x-specific information has been removed from this guide, as it has been incomplete and misleading.

1.1 chroot jail

While we're preparing the binaries for miniBSD, we'll be overwriting some binaries of the base system with our custom, net4501-tuned binaries (for the sake of simplicity we don't install them directly into our net4501 directory, but install them into the base system and copy them over later). If you do not want to harm/change your base system in any way, you have two choices: a chroot jail or a second hard disk. The latter is easier - you just install FreeBSD onto a separate hard disk (or at least a separate partition) and use it just for preparing miniBSD, and nothing else. The other method allows you to work in a "sandbox" without requiring a second hard disk or partition.

A chroot jail is pretty easy to set up - you just make a directory on a partition with enough space (800 MB recommended), say, /usr/jail. Then you put the FreeBSD installation CD-ROM in your drive, start up /stand/sysinstall, begin a custom installation and set the "Install Root" in the options to point to your jail directory. After that, you start the installation and when it's done you will have a "FreeBSD inside FreeBSD". Type

```
chroot /usr/jail
```

and you will find that the file system root has changed - what you now see as the root (/) in your shell is really only /usr/jail. As long as you do not leave that shell, any modifications you make will only take place in /usr/jail and its subdirectories. You can make world or whatever you please, and it will not affect your base system.

Don't forget that if you leave the shell, you'll have to chroot again.

Contributed by Murray Taylor: modify the "set prompt" line in /usr/jail/root/.cshrc if you want to be reminded of the fact that you're inside the jail by the shell prompt:

```
set prompt = "MiniBSD %~ %# "
```

You may also want to copy /etc/localtime to /usr/jail/etc/localtime in order for timestamps etc. to be displayed correctly.

---

2. Making the directory tree

We'll put together our whole miniBSD installation in one directory, so in the end we could theoretically just copy everything from that directory over to the flash card. I'll leave it up to you where you're going to create this directory; mine is `/usr/minibsd` and the rest of the guide assumes that yours is, too (just make sure you got about 100 MB of space left on the partition where you create it). That done, we need to populate this directory with some standard subdirectories (bin, dev, etc, usr, and so on). You could use mtree to populate it with empty directories, but I prefer to do it manually. Here is the tree of directories you have to mkdir:

```
.
|-- bin
|-- boot
|   `-- defaults
|-- dev
|-- etc
|   |-- defaults
|   |-- mtree
|   |-- namedb
|   |-- ppp
|   |-- ssh
|   `-- ssl
|-- mnt
|-- modules
|-- proc
|-- root
|-- sbin
|-- tmp -> /var/tmp
|-- usr
|   |-- bin
|   |-- lib
|       `-- aout
|   |-- libexec
|   |-- local
|   |-- sbin
|   `-- share
|       `-- misc
`-- var
```

Permissions are 0755 (root.wheel) for all directories except `/proc` (0555). You may also want to `chmod 0700 /root`, but that is a matter of personal taste. :)
Don't forget the link from `/tmp` to `/var/tmp` - it's important as the root file system will later be mounted read-only (we'll provide a memory file system for /var).

---

3. Rebuilding the boot loader

The net4501 BIOS seems to have a little bug in its serial console emulation - if you boot a standard FreeBSD bootloader on it, it will spread the characters (e.g. the "swoosh" (- \ | / - ...) all over your terminal. When the kernel boots, things are normal again, but it may be important to see the error messages in case the bootloader cannot boot the kernel. To fix this, we have to recompile the boot loader (don't worry, it's easy and quick).

Note: you'll need to have the developer distribution installed on your base system to recompile the boot loader – just the kernel sources is not sufficient. You can install the developer distribution with /stand/sysinstall. If you can't/don't want to recompile the boot loader, you can download a precompiled boot loader (for 4.7-RELEASE) here. gunzip it and throw it in /boot/loader. Don't forget to chmod 555 it.

Edit `/sys/boot/i386/libi386/Makefile` and *comment out* the line that reads

```
CFLAGS+= -DTERM_EMU
```

Leave the editor, cd to `/sys/boot` and type `make clean && make && make install` (of course this will also change the boot loader of your base FreeBSD system to not use terminal emulation, but that doesn't hurt much).

---

4. Dynamic vs. static executables

In a normal FreeBSD installation, the binaries in /bin and /sbin are linked statically, which means they do not depend on any libraries in order to work. This makes them much bigger, though, and you can save several MBs (about 6.7 MB with my binary set) by linking them dynamically, too. Since we keep everything on one partition, they would still be usable in single-user mode because the libraries are available. The only disadvantage is that if your libraries get corrupted, you won't be able to do much on the system to fix it. Decide for yourself whether you want them to be dynamically linked. If you do, edit /etc/make.conf and add the line

```
NOSHARED=no
```

to it. Then cd to `/usr/src/bin` and type `make clean && make && make install`. Do the same for /usr/src/sbin. Voilà.

NOTE: if you actually run "make install" with NOSHARED=no set on a real FreeBSD system (not in a jail as explained in chapter 1.1), you should really make sure that your /usr directory is in your root partition (not in a separate partition). Otherwise your next reboot will fail horribly because /sbin/init is linked dynamically but the required libraries can't be found (since the other partitions are not mounted at that time, yet).

If you feel like saving another 400 KB (with my binary set), you can do the same for /usr/src/usr.bin, because a few binaries in there (bzip2, gzip and tar among others) are also linked statically by default.

---

5. Copying the binaries over

Now, for the fun part of it... You can now go visit your /bin, /sbin, /usr/bin, /usr/sbin, /usr/libexec directories and pick out the binaries you want and copy them over in the respective directories in /usr/minibsd. Just kidding. ;) I can save you most of the hard work. I made a little perl script that reads a file which lists all the necessary binaries and copies them over, recreating the proper hard links in the process (for example, /bin/ln and /bin/link are the same file, but if you copy them over with cp, you end up with two copies of ln/link on your miniBSD file system, thus wasting a lot of space).

Get mkmini.pl and my list of binaries, minibsd.files.

Now execute mkmini.pl (make sure your /usr/minibsd/bin, /usr/minibsd/usr/bin, ... directories are empty, otherwise mkmini.pl may fail while trying to make the links):

```
perl mkmini.pl minibsd.files / /usr/minibsd
```

mkmini.pl will read the list of files in minibsd.files and copy them from your system in / into your miniBSD root file system in /usr/minibsd. My file list contains a reasonable amount of binaries tuned to sacrifice the least amount of functionality for my purpose, but still fit well on a CF card. Things like perl and sendmail are missing, however, because I don't really need them. If you do, change minibsd.files to reflect your requirements.

If you feel like saving almost 200 KB, you can copy the stripped-down termcap file from PicoBSD over the standard /usr/minibsd/usr/share/misc/termcap file. You'll find the replacement file in /usr/src/release/picobsd/mfs_tree/etc/termcap. The standard termcap file contains many terminal types that you probably won't use anyway. There's also a stripped services file in /usr/src/release/picobsd/mfs_tree/etc/services. *(suggested by Stuart Henderson)*

---

6. Making the kernel

You should make a custom kernel to save some more space and to include options specific to the net4501 (you need to have sources/compilers installed in your base system). Go to /sys/i386/conf, make a copy of GENERIC and edit it. You can most probably throw out things like SCSI controllers, NICs you don't need, USB support etc. I even removed things like CD9660 since I can't connect a CD-ROM drive to my SBC anyway. If you want any CPU optimizations, edit /etc/make.conf now (I used CPUTYPE=i486 since the net4501 is 486-based). Finally, add the following line to your kernel configuration file:

```
options CLK_USE_I8254_CALIBRATION
```

This is needed because the AMD Elan SC520 CPU used in the net4501 runs its clock at 1.1892 MHz instead of the standard 1.19318 MHz. If you don't use it, your clock will probably be slow by several minutes per day (Thanks to Matt Peterson for pointing this out!).

In FreeBSD 4.7, it is also suggested to add the following kernel options:

```
options CPU_ELAN
options HZ=250
```

This will allow the Elan CPU's general purpose timer #2 to be used as a timecounter, which is a vast improvement over using the i8254 timer. HZ has to be bumped from 100 to 250, though. It also enables the /dev/elan-mmcr device for access to the Elan's MMCR registers.

Since the net4501 does not have real video/keyboard hardware, the kernel will fall back to the serial console. The default settings are 9600 bps – if you want to change that, you'll have to add an `option CONSPEED=x` to your kernel config file. You should make sure that the baud rate set in the comBIOS and in the kernel config agree. I just set the comBIOS speed to 9600, so no kernel config settings are required. There is an option in the comBIOS to prevent the kernel from changing the console baud rate; I have not verified that it works, however.
Note that since the boot loader uses the BIOS to output its messages, we don't have to care about it because the comBIOS will automatically redirect the output to the serial console.

Now do

```
config MYKERNEL
```

(of course replacing MYKERNEL with the name of the config file you just created)

and cd to /sys/compile/MYKERNEL. Do a

```
make depend && make
```

When it's done, you should have your kernel. We will now install it gzipped to save even more space:

```
gzip -9 kernel
cp kernel.gz /usr/minibsd
```

The FreeBSD loader will automatically find the .gz kernel there and gunzip it prior to invoking it. Note that you could also use kgzip, however you'd lose all symbols in the process and tools like top would break. Since we don't care about the additional 300 KB that the loader requires, we do it this way.

If you need any modules (I need some netgraph modules for PPPoE support), you can simply copy them over into your /usr/minibsd/modules. You'll find them in `/sys/compile/MYKERNEL/modules/usr/src/sys/modules/[modulename]/*.ko`. Just BTW, for PPPoE client support you'll need netgraph.ko, ng_ether.ko, ng_pppoe.ko and ng_socket.ko.

---

7. Copying the libraries

We did not copy all the essential libraries in /usr/lib yet. These are not listed in minibsd.files because the list of required libraries depends on which binaries you have installed. In an earlier version of this guide, we would just copy all libraries in /usr/lib and then remove those that we were sure would not be required (e.g. *.a). That still left a lot of unnecessary libraries (about 5.9 MB with FreeBSD 4.6) lying around. If you use my file list (minibsd.files), the libraries that are absolutely necessary weigh in at 3.2 MB - so that means another 2.7 MB saved.

I wrote a script, mklibs.pl, which walks through your miniBSD tree and uses ldd on all binaries to extract a list of only those libraries that are required for at least one binary in your miniBSD system.

Get mklibs.pl now and run it:

```
perl mklibs.pl /usr/minibsd > minibsd.libs
```

Take a look at the output (minibsd.libs). If it looks OK, you can in turn use it with mkmini.pl:

```
perl mkmini.pl minibsd.libs / /usr/minibsd
```

The only libraries that are still missing now are the PAM modules. Doing a

```
cp -p /usr/lib/pam* /usr/minibsd/usr/lib
```

should fix that. If you don't want all PAM modules (you probably won't need Kerberos modules, for example), you should copy at least pam_deny, pam_permit, pam_unix, pam_login_access and pam_nologin. Finally, if you `mkdir /usr/minibsd/usr/lib/aout`, you will avoid the warning message that appears otherwise.

That's it. Our libraries directory has been populated.

---

8. Populating /etc

Of course we also need configuration files for our miniBSD... Your best bet is probably to visit your existing /etc and copy all the files/directories you need to /usr/minibsd/etc. As a starting point, here's my tree of /usr/minibsd/etc (files which you'll likely have to modify are marked red):

```
/usr/minibsd/etc
|-- auth.conf
|-- crontab
|-- defaults
|   |-- make.conf
|   `-- rc.conf
|-- dhclient.conf
|-- disktab
|-- fbtab
|-- fstab
|-- ftpusers
|-- gettytab
|-- group
|-- host.conf
|-- hosts
|-- hosts.allow
|-- hosts.equiv
|-- hosts.lpd
|-- inetd.conf
|-- localtime
|-- login.access
|-- login.conf
|-- master.passwd
|-- motd
|-- mtree
|   |-- BSD.include.dist
|   |-- BSD.local.dist
|   |-- BSD.root.dist
|   |-- BSD.sendmail.dist
|   |-- BSD.usr.dist
|   |-- BSD.var.dist
|   |-- BSD.x11-4.dist
|   `-- BSD.x11.dist
|-- namedb
|   |-- PROTO.localhost.rev
|   |-- make-localhost
|   |-- named.conf
|   `-- named.root
|-- networks
|-- newsyslog.conf
|-- objformat
|-- pam.conf
|-- passwd
|-- ppp
|   `-- ppp.conf
|-- profile
|-- protocols
|-- pwd.db
|-- rc
|-- rc.conf
|-- rc.diskless1
|-- rc.diskless2
|-- rc.firewall
|-- rc.i386
|-- rc.local
|-- rc.network
|-- rc.serial
|-- rc.shutdown
|-- rc.sysctl
|-- resolv.conf
|-- security
|-- services
|-- shells
|-- skeykeys
|-- spwd.db
|-- ssh
|   |-- primes
|   |-- ssh_config
|   |-- ssh_host_dsa_key
|   |-- ssh_host_dsa_key.pub
|   |-- ssh_host_key
|   |-- ssh_host_key.pub
|   `-- sshd_config
|-- ssl
|   `-- openssl.cnf
|-- sysctl.conf
|-- syslog.conf
|-- termcap -> /usr/share/misc/termcap
|-- ttys
`-- wall_cmos_clock
```

/etc/periodic is missing here - I find that I can make do without it on my SBC. Check if there's anything in your crontab that needs to be removed (in this case, the 'periodic' jobs have to be removed, too).

Edit fstab and remove all mounts you don't need - you'll probably be left with an fstab that looks about like this:

```
/dev/ad0a       /          ufs     ro      1   1
proc           /proc      procfs  rw      0   0
```

I have my root file system on ad0a since ad0 is the device that CF cards get assigned to on the net4501. Don't forget to change the 'rw' options of your root file sytem to 'ro' - we want the root FS to be mounted read-only. This is necessary since flash media has a limited number of write cycles until it wears out. If you keep it mounted read-write and have log files or something like that on it that gets written periodically, your flash card won't live long (so they say).

The next file we have to look at is rc.conf. You'll most likely want to disable daemons like usbd or sendmail. My rc.conf looks like this:

```
hostname="fb.neon1.net"
ifconfig_sis0="192.168.0.200 netmask 255.255.255.0"
kern_securelevel_enable="NO"
nfs_reserved_port_only="YES"
sendmail_enable="NONE"
sshd_enable="YES"
usbd_enable="NO"
inetd_enable="NO"
portmap_enable="NO"
diskless_mount="/etc/rc.diskless2"
update_motd="NO"
varsize=8192
```

diskless_mount instructs the boot script, /etc/rc, to invoke /etc/rc.diskless2, which is supplied with FreeBSD by default and serves to mount memory file systems for /var and /dev. varsize sets the size of that file system to 4 MB (8192 blocks) - you may have to change that if you have big log files in there. update_motd instructs rc not to try to update the contents of /etc/motd with the current system release name - it won't be able to do that, since / is mounted read-only.

Don't forget to edit /etc/newsyslog.conf to rotate log files more often and don't archive them. You probably won't want much logging anyway, since all your log files will be lost when you reboot your machine.

If you're running dhclient (i.e. if you want your miniBSD machine to act as a DHCP client), remember that dhclient tries to write nameserver information obtained via DHCP to the file /etc/resolv.conf. This is not normally possible, of course, because the root filesystem is read-only. The solution is to symlink /etc/resolv.conf to /tmp/resolv.conf. Thanks to Roddy Collins who pointed this out!

Of course there are other modifications you may have to make to render your system useful, but as I said, these are documented elsewhere and I don't intend to reinvent the wheel.

Don't forget that if you make any changes to master.passwd, you'll have to rebuild the database with

```
pwd_mkdb -p -d /usr/minibsd/etc /usr/minibsd/etc/master.passwd
```

If you feel like saving some extra memory, open rc.diskless2 and change the line that reads

```
mount_md 4096 /dev 3 512
```

to

```
mount_md 1024 /dev 3 512
```

...I find that 512 KB is enough for my device files. :) You may be able to reduce that figure even more, but be sure that all device files fit on it! (`df` is your friend)

If you want to get rid of the 10 second boot delay, edit /usr/minibsd/boot/loader.rc and append the line

```
autoboot 0
```

to it.

If you want to be able to actually log into your net4501 via the serial console (not just view the loader/kernel/boot messages), edit /etc/ttys and change the line that reads

```
ttyd0 "/usr/libexec/getty std.9600" dialup off secure
```

to

```
ttyd0 "/usr/libexec/getty std.9600" vt100 on secure
```

Change the speed, if necessary.

---

9. /dev fun

To boot and run your miniBSD system, you also need some device special files. The FS on which they reside has to be mounted read-write in order for all the stuff like getty to work, so rc.diskless2 builds a memory FS for them. The kernel will still expect to find the device file of the root partition to mount inside the root partition, so we'll build a complete /dev on our miniBSD root FS. Start by doing

```
cp /dev/MAKEDEV /usr/minibsd/dev
cd /usr/minibsd/dev
sh MAKEDEV all
```

Make sure you have the device file for your miniBSD root partition! (use `sh MAKEDEV [name]` if you don't). If you feel like saving a few extra KBs of space, check the contents of /usr/minibsd/dev and delete everything you don't need (for example, da* SCSI disk files, ad[1-3]*, and so on).

rc.diskless2 expects to find a file named /conf/dev.cpio.gz with all the device files - if it can't, it will find(1) all device files in your root partition and generate the cpio file in /tmp. This can take a "long" time, so we'll pre-create that file:

```
cd /usr/minibsd
mkdir conf
find -x dev | cpio --create -H newc | gzip > conf/dev.cpio.gz
```

That's it. We now have two copies of our device files - one in /dev on the root partition, and one in the dev.cpio.gz file mentioned above. When your system boots, the kernel will find the expected device files on the root FS, then rc.diskless2 will mount a memory file system and populate it with the same device files so /dev is read-writeable for later.

---

10. Tar'ing everything up

Because of the aforementioned inability of cp to reproduce hard links, we'll build a tar archive of our whole miniBSD installation. Do the following:

```
cd /usr/minibsd
tar cfvz /usr/minibsd-46-1.tgz *
```

Of course, the name of the .tgz file is a matter of personal taste. :)
When it's done, you'll be left with a .tgz file that contains everything (well, almost) for your miniBSD.

---

11. Generating the flash image

Although we could use the CF card like a hard disk, disklabel and newfs it and explode our tar archive directly on it, I feel it's easier and faster to do that task in a disk image and dd that directly onto the flash card - besides, it prolongs the life of the CF card by writing every sector only once.

We'll use vnconfig to get a virtual disk that we can disklabel. You will have to know the size of your flash card in sectors (512 byte units), but you cannot figure this out without actually connecting the card to your machine (see the beginning of the next section for information on how to do this). Once you got it connected, use the following command to find out the number of sectors:

```
disklabel -rwn ad[n] auto | grep sectors/unit
```

...of course substituting the proper device number of your flash card for [n].

Hint: if disklabel reports errors like "No space left on device", try the following command first:
`dd if=/dev/zero of=/dev/rad[n] bs=1k count=20`
This will erase any partition information that may already be present on the CF card.

Note: you may have to create the device first by doing `cd /dev; sh MAKEDEV ad[n]`.

Now we'll create a disk image of the same size, initially filled with zeroes:

```
dd if=/dev/zero of=/usr/minibsd-disk.bin bs=512 count=[number of sectors on
your flash card]
```

Let's use this disk image file as a vn device so we can disklabel it:

```
vnconfig -s labels -c vn0 /usr/minibsd-disk.bin
```

Our new virtual disk has to be partitioned and the file system created. Use:

```
disklabel -Brw vn0 auto
disklabel -e vn0
```

You'll be dropped into your editor. Find the line that starts with c: and **duplicate it**, changing the c: to a: and the fstype to `4.2BSD`. This is our root partition, which will span over the whole slice. If you want a second partition (e.g. for configuration information), reduce the size of the a: slice by the amount of space you want for your second partition, then make another line starting with e:, an offset equal to the size of a:, and a size equal to what you subtracted from a:. Save.

Note that we don't have a swap partition - there's nowhere to put it. You can't put swap partitions on your flash card - it's way too slow and would soon ruin your card. So you better make sure that you've got enough memory for your applications...

Now let's newfs and mount our partition:

```
newfs -b 8192 -f 1024 -U /dev/vn0a
mount /dev/vn0a /mnt
```

(if you made a second partition earlier on, don't forget to newfs that one, too). So we got our virtual disk mounted on /mnt, and we're now ready to explode our .tgz onto it:

```
cd /mnt
tar xfvzP /usr/minibsd-46-1.tgz
```

That's it. Let's clean up:

```
cd /
umount /mnt
vnconfig -u vn0
```

Note: if, after completing the next section, you find that you can't boot off the CF card, there might be a C/H/S mismatch problem. See this message from Jeremy Cooper for details and a solution.

---

12. Copying everything to the flash card

This is a bit tricky. You could netboot (with PXE) your net4501 with the CF card installed and copy everything over the network, but preparing a bootable FreeBSD for this purpose isn't quite simple, either. You'll need some kind of adapter in order to connect your CF card to your base system. If your base system is a notebook, you can use a CF -> PCMCIA adapter (enable pccardd and you'll see the card as an ATAPI device, like for example ad8) or a CF -> IDE adapter if it's a desktop machine. Either way, you'll end up with an ad[n] device that represents the CF card (it behaves like an IDE hard disk).

If you don't yet have the device files for your ad[n] device in the /dev of your base FreeBSD installation, cd to /dev and `sh MAKEDEV ad[n]` now.

We can now simply transfer the disk image we created in the previous step block-by-block onto our flash card:

```
dd if=/usr/minibsd-disk.bin of=/dev/rad[n] bs=8k
```

(make sure you get the value for [n] right in this command or you might erase your hard disk!!!)

If you want to see the progress, type Ctrl-T.

---

13. Booting the new system

Shut down your base system, take out the flash card and put it in your net4501. Connect its console port to some PC and run a terminal emulator. You're now ready to boot your new miniBSD system - power up your net4501. Don't panic if it won't boot right away - read the error messages, most probably it's only a file missing somewhere.

If you see warnings like the following:

```
Warning: Block size restricts cylinders per group to xx.
```

you can safely ignore them.

Once you've got it up and running, you can start enabling the various daemons you need, configure ipfw, ppp, natd, whatever. See the next section for help on installing ports.

If you need to read-write-mount the root file system, use

```
mount -uw -o noatime /
```

Don't forget to do

```
mount -ur /
```

when you're done!

The noatime flag indicates that the atime (time of last access) should not be updated. Without noatime, every file access (even if it's only a read) will result in a write to the CF card, causing it to wear out more quickly. atime is not normally needed.

WARNING: some people believe that remounting a writeable file system read-only again could lead to file system corruption in the long run. I don't know if it's true or not, but I would not rule out that possibility. At least it has happened to me once that I could not remount ro without using the -f (force) flag (a very bad thing to do) because some daemon process had opened a file read-write in the meantime (though fstat would not tell me about it). So I guess the more secure way would be to do a clean reboot after mounting a file system read-write.

If you make any changes on your running net4501, I suggest making them also on your base FreeBSD system (in your /usr/minibsd or whatever you chose). This will make things a lot easier when it's time to upgrade to the next release of FreeBSD. :)

Note that it's a good idea to generate new ssh keys now, since otherwise your net4501 would share the same set of keys with your base system:

```
ssh-keygen -t rsa1 -N "" -f /etc/ssh/ssh_host_key
ssh-keygen -t rsa -N "" -f /etc/ssh/ssh_host_rsa_key
ssh-keygen -t dsa -N "" -f /etc/ssh/ssh_host_dsa_key
```

Appendix A - Installing ports

You'll most likely want to install some ports to get your system to do something useful. You can just boot your base system, then build the port as usual but specify PREFIX=/usr/minibsd/usr/local while make install'ing. If you want to do it extra-cleanly, install the port in your base system, then figure out which binaries/libraries/configuration files/rc.d scripts you REALLY need and copy them over to their respective position in /usr/minibsd.
Contributed by Tom Thompson: you can use `pkg_info -qxL [package name]` *to help finding out which files were installed by a particular port.*

Appendix B - But I want my perl!

OK, OK... It will take about an additional 10 MBs (without pod). Add the line

```
usr/bin/perl:usr/bin/perl5:usr/bin/perl5.00503
```

to your minibsd.files. Copy over /usr/libdata/perl (you can omit /usr/libdata/perl/pod). Make sure you get the permissions right (tar'ing it up and using tar with -P while extracting is the easiest way).

Appendix C - Saving data across a reboot

If you run something like dhcpd on your net4501, you'll find that your leases file is lost every time you reboot/power cycle your net4501. I made a small separate partition on my flash card just for the purpose of saving the contents of /var/db before shutting down. Just add the following lines to your rc.shutdown, just where it says "Insert other shutdown procedures here":

```
# Save databases
echo -n 'Saving databases in /var:'
mount -u -o rw /conf
rm -Rf /conf/varsave/db
cp -Rp /var/db /conf/varsave
mount -u -o ro /conf
echo '.'
```

You need to have an fstab entry for /conf, and the directory /conf/varsave must exist. Add the following to your rc.local (create it if it doesn't exist) to restore the databases on boot:

```
# Load saved databases
cp -Rp /conf/varsave/db /var
```

Note that rc.shutdown is only invoked if you shut your system down with shutdown(8), not with halt/reboot(8). Use `shutdown -h now` instead.
