Network Boot and Exotic Root HOWTO

Brieuc Jeunhomme

Logilab S.A.

Revision History

Revision 0.2.2	2001−12−08	Revised by: dcm
Licensed GFDL		
Revision 0.2.1	2001−05−21	Revised by: logilab
Fixed bibliography and artheader		
Revision 0.2	2001−05−19	Revised by: bej
Many improvements and included Ken Yap's feedback.		
Revision 0.1.1	2001−04−09	Revised by: logilab
First public draft.		
Revision 0.1	2000−12−09	Revised by: bej
Initial draft.		

This document explains how to quickly setup a linux server to provide what diskless linux clients require to get up and running, using an IP network. It includes data and partly rewritten text from the Diskless−HOWTO, the Diskless−root−NFS−HOWTO, the linux kernel documentation, the etherboot project's documentation, the linux terminal server project's homepage, and the author's personal experience, acquired when working for Logilab. Eventually this document may end up deprecating the Diskless−HOWTO and Diskless−root−NFS−HOWTO. Please note that you'll also find useful information in the From−PowerUp−to−bash−prompt−HOWTO and the Thin−Client−HOWTO, and the Claus−Justus Heine's page about NFS swapping.

Table of Contents

Table of Contents

1. Introduction

1.1. What is this all about?

Recent linux kernels offer the possibility to boot a linux box entirely from network, by loading its kernel and root filesystem from a server. In that case, the client may use several ways to get the first instructions it has to execute when booting: home made eproms, special network cards implementing the RARP, BOOTP or DHCP protocols, cdroms, or bootloaders loaded from a boot floppy or a local hard drive.

1.2. Thanks

Logilab sponsored this HOWTO. Check their website for new versions of this document. I also thank the etherboot and linux terminal server project developers and webmasters, who made it really possible to boot a Linux worstation over a network.

Very special thanks go to Ken Yap, member of the etherboot project, whose comments greatly helped to improve the quality of this document.

1.3. Diskless booting advocacy

1.3.1. Buying is cheaper than building

Sometimes, buying a diskless linux computer will be cheaper than building! Checkout the list of commercial sites given in appendix, which are selling diskless linux network cards and diskless computers. These companies do mass production of linux diskless computers selling millions of units and thereby reducing the cost per unit.

1.3.2. Advantages of diskless computers

Diskless computers will become more and more popular in the next years. They will be very successful because of the availability of very high−speed network cards at very low prices. Today 100 Megabit per second (12.5 MB per sec transfer rate) network cards are common and in about 1 to 2 years 1000 MBit (125 MB per sec transfer rate) network cards will become very cheap and will be the standard.

In near future, monitor manufacturers will place the CPU, NIC, RAM right inside the monitor to form a diskless computer. This eliminates the diskless computer box and saves space. The monitor will have outlet for mouse, keyboard, network RJ45 and power supply.

The following are benefits of using diskless computers:

- Total cost of ownership is very low in case of diskless computers. Total cost of ownership is cost of initial purchasing + cost of maintenance. The cost of maintenance is usually 3 to 5 times the cost of initial computer purchase and this cost is recurring year after year. In case of diskless computers, the cost of maintenance is completely eliminated.

- All the backups are centralized at one single main server.
- No need of UPS battery, air−conditioning, dust proof environment for diskless clients, only server needs UPS battery, A/C and dust proof environment.
- A better protection from virus attack − Some computer virus cannot attack diskless computers as they do not have any hard disk. This kind of virus cannot do any damage to diskless computers. Only one single server box needs to be protected against virus attack. This saves millions of dollars for the company by avoiding installation of vaccines and cleaning the hard disks.
- Servers can have large powerful/high performance hard disks, can optimize the usage of disk space via sharing by many diskless computer users. Fault tolerance of hard disk failure is possible by using RAID on main server.
- On some installations: sharing of central server RAM memory by many diskless computer users. For example, if many users are running a web browser remotely on a server, then there will be only one copy of this web browser in its RAM.
- Very few system administrators required to maintain central server.
- Zero administration at diskless client side. Diskless computers are absolutely maintenance free and troublefree.
- Long life of diskless clients.
- Eliminates install/upgrade of hardware, software on diskless client side.
- Eliminates cost of cdrom, floppy, tape drive, modem, UPS battery, printer parallel ports, serial ports etc...
- Can operate in places like factory floor where a hard disk might be too fragile.

1.4. Requirements

1.4.1. Hardware requirements

The most important thing in order to boot from network is to have an equipment which enables the stations to execute a bootloader, which will get the kernel on the server and launch it. Another solution is to use a device which will load a local kernel, which will mount the root filesystem on the server. There are several solutions: home made eproms containing the first instructions to execute when booting the station, boot with BOOTP/DHCP capable network adapters, or a local floppy, a tiny hard drive, or a cdrom to load the kernel. Note that some vendors also sell network booting capable stations: for instance, some Sun stations implement the BOOTP protocol.

Other hardware requirements depend on the configuration you plan to use: on some sites, every application run by the stations is executed remotely on the server, this implies that a very high−performance server is required, but only light stations are required: depending on what they will have to do, 80486 CPUs with 16 MB of RAM may be enough. On the other side, if application programs are really executed locally on the stations, the requirements for the stations depend completely on these applications. In that case, only a small server is required. A 80486 CPU with 32 MB of RAM will be sufficient for a small number of stations, but more memory will be necessary in very large installations with hundreds or thousands of machines. Note the server's CPU does not really matter for such an installation.

1.4.2. Software requirements

Linux kernel version 2.0 or above sources are required. All tools required to build a linux kernel are also necessary (see the linux kernel documentation for more information on this).

A BOOTP daemon (a DHCP daemon may also do fine, but I won't explain how to configure this), a NFS daemon (if you want to mount the root filesystem on a remote server), are also required. We will also need a TFTP daemon if you plan to load the kernel remotely. At last, the mknbi utility provided with the etherboot distribution, and, if you use LanWorks EPROMs, like those included in the 3c905 3com ethernet adapter, you will also need the imggen utility, available at ftp://www.ltsp.org/pub/download/lts/contrib.

1.5. Acknowledgements and related documentation

This documentation has been written for experimented system administrators, who are already aware of linux fundamentals, like the use of grep, sed, and awk, basic shell programming, the init process and the boot scripts, kernel compilation, and NFS server configuration. Experience of kernel arguments passing should also help. Information on these subjects can be found respectively in the grep, sed, awk, and bash man/info pages, in the Bootdisk−HOWTO, the From−PowerUp−To−Bash−Prompt−HOWTO, the Kernel−HOWTO, the BootPrompt−HOWTO, the bootparam man page, the rdev man page, the NFS−HOWTO, and the exports manual page.

There are many sources of information on network booting, but, and this is why I wrote this HOWTO, none describes all the existing ways of booting over a network, and much of them are specific to a way of operating. The most useful to me has been the documentation provided by the linux terminal server project, although I did not use the packages they recommend, and I have chosen to describe here how to proceed without these packages, because they configure things so that every application program is executed remotely on a server. Useful information can also be found on the etherboot project's homepage.

At last, you can also find useful but succinct information in the kernel's source tree, in `/usr/src/linux/Documentation`, assuming your kernel source tree resides in `/usr/src/linux`.

1.6. Feedback

I will highly appreciate any feedback about this document. Please feel free to mail me at `<bbp@via.ecp.fr>` if you have any comment, correction, or suggestion. You may also use `<contact@logilab.fr>`.

1.7. Copyright Information

2. Diskless booting operation overview

Hey, you think it's time to start with the real stuff, right? Here we go.

2.1. Obtaining IP parameters

One could wonder how a station may boot over an IP network if it doesn't even know its own IP address. In fact, three protocols enable the client to obtain this information and some additional configuration parameters:

- RARP: this is the simplest of these protocols. However I guess it does not enable the server to specify how the client should download the kernel, so we won't use it (In fact, there is a convention that uses the IP address of the workstation as filename, e.g. a client getting the address 192.168.42.12 by RARP might ask for `/tftpboot/192.168.42.12` by TFTP, as the linux kernel does. The filename might also be the hex form of the IP address, this is implementation dependant, and is not mandatory.).
- BOOTP: this protocol allows a server to provide the client (identified by its hardware MAC address) with much information, in particular its IP address, subnet mask, broadcast address, network address, gateway address, host name, and kernel loading path. This is the one we will use.
- DHCP: this is an extension of BOOTP.

2.2. Loading the kernel

When the client has got its IP parameters, if the kernel is not on a local support (like a floppy, a cdrom, or a hard drive), the client will start to download it via TFTP. Its location is given by the BOOTP/DHCP server. A server (not necessarily the BOOTP/DHCP server) will also have to run a TFTP daemon for non local kernels. The kernel one obtains after compilation can not be used "as is" for BOOTP/DHCP operation, its binary image has to be modified with the **mknbi** utility (and then modified again with the **imggen** utility if you use LanWorks EPROMs). The **mknbi** utility should also be used to modify kernels that will be written in a ROM.

2.3. Mounting the root filesystem

After the kernel has started, it will try to mount its root filesystem. The location of this filesystem is also obtained through BOOTP/DHCP, and it is mounted via NFS. It means a client may use BOOTP twice for booting: the first time to get its kernel, and the second time to learn the location of the root filesystem (which may be on a third server).

Another solution is to use a ramdisk as root filesystem. In this case, the ramdisk image is obtained with the kernel via TFTP.

2.4. Terminating the boot process

When the root filesystem is mounted, you can start breathing: you can at least use your swiss army knife with its sh, sed, and awk blades. In fact, you will have to customize the initialization scripts of the client's filesystem: for instance, you will have to remove all hard drive, floppy or cdrom related stuff from `/etc/fstab` (when your stations are not equipped with these devices), you may also have to inhibit swap partitions activation (note there is a way to swap over NFS or network block devices). You also will have to automagically generate all network configuration files at boot time if several clients use the same remote root filesystem.

3. Building the kernel

Firs of all, build a kernel for the clients. I suggest you build it on the server, this will be useful later for modules installation. Use a zImage to reduce its size. Include everything you need, but try to use as many modules as possible, because many BOOTP client implementations are unable to load very large kernels (at least on intel x86 architectures). Also include iramdisk support, NFS protocol support, root filesystem on NFS support, support for your NIC, kernel level IP autoconfiguration via BOOTP; *do not use modules for these!* Then, if you plan to use the same remote root filesystem for several clients, add support for ext2fs or some other filesystem and ramdisks (16 Megabytes ramdisks will do fine on most systems). You can then modify the kernel arguments as usual (see the BootPrompt−HOWTO for information on this topic), but you will have another opportunity to modify kernel arguments later.

Then, if you plan to use BOOTP, copy the kernel zImage on the server. We will assume it resides in `/tftpboot`, its name is zImage, the name of the image you want to create from this zImage for BOOTP operation is kernel, and the nfs root filesystem will reside in `/nfsroot`.

Issue the following commands on the server (the mknbi package should be installed):

```
# cd /tftpboot
      # chmod 0555 zImage
      # chown root:root zImage
      # mknbi-linux zImage --output=kernel --rootdir=/nfsroot
```

If you are using LanWorks EPROMs, also issue the following commands (you need the imggen utility):

```
# mv −f kernel tmpkernel
      # imggen −a tmpkernel kernel
      # rm −f tmpkernel
```

Your kernel is ready for BOOTP/DHCP/ROM operation. You of course don't need to do this if you plan to use a local drive.

3.1. When the root filesystem is on a ramdisk

It is possible to use a ramdisk for the root filesystem. In this case, the command used to modify the kernel's binary image is slightly different. If you choose to do so, you have to enable support for initial ramdisk (initrd), and you probably don't need NFS support, or you probably can compile it as a module.

Its time to give an overview of what happens when you use initrd. The full documentation for this is in your kernel source tree, in the `Documentation/initrd.txt` file. I have to warn you I did never try this :).

When initrd is enabled, the boot loader first loads the kernel and the inital ramdisk into memory. Then, the ramdisk is mounted read−write as root filesystem. The kernel looks for a `/linuxrc` file (a binary executable or a script beginning with #!). When `/linuxrc` terminates, the traditionnal root filesystem is mounted as `/`, and the usual boot sequence is performed. So, if you want to run your box entirely from ramdisk, you just have to create a link from `/linuxrc` to `/sbin/init`, or to write there a shell script to perform any action you like, and then shutdown the computer.

After the kernel has been compiled, you have to build a root filesystem for your installation. This is explained in the "Clients setup, creation of the root filesystem" section. I will assume here that this is already done and that the root filesystem for your clients temporarily resides in `/tmp/rootfs`. You now have to create a ramdisk image. A simple way to do so is the following:

- Make sure the computer you are working on has support for ramdisks and has such a device (`/dev/ram0`).
- Create an empty filesystem with the appropriate size on this ramdisk:
  ```
  # mke2fs −m0 /dev/ram0 300
  ```
- Mount it somewhere:
  ```
  # mount −t ext2 /dev/ram0 /mnt
  ```
- Copy what you need for your new root filesystem, and create your future `/linuxrc` if you did not create it in `/tmp/rootfs/linuxrc`:
  ```
  # cp −a /tmp/rootfs/* /mnt
  ```
- Unmount the ramdisk:
  ```
  # umount /mnt
  ```
- Save the ramdisk image to some file and free it:
  ```
  # dd if=/dev/ram0 of=initrd bs=1024 count=300
        # freeramdisk /dev/ram0
  ```

What was toled above about LanWorks PROMs is also true if you use initrd.

Then, you have to modify the kernel image, as was told above, with the **mknbi−linux** utility. Its invocation will slightly differ from the above, though (I will assume your just compiled zImage resides in `/tftpboot/zImage` and your initial ramdisk image resides in `/tmp/initrd`):

```
# cd /tftpboot
        # chmod 0555 zImage
        # chown root:root zImage
# rdev zImage /dev/ram0
        # mknbi−linux zImage −−output=kernel −−rootdir=/dev/ram0 /tmp/initrd
```

4. Daemons setup

4.1. NFS daemon

Just export the directory in which the client's root filesystem will reside (see the exports manpage for more information about this topic). The simplest is to export it no_root_squash and rw, but a perfect setup would export most of the root filesystem root_squash and ro, and have separate lines in the `/etc/exports` for directories which really require no_root_squash and/or rw. Just start with everything rw and no_root_squash, the fine tuning will be done later.

Of course, you don't need any NFS server at all if you plan to run your clients entirely from ramdisk.

4.2. BOOTP daemon

I assume you have installed the bootpd package. The default configuration file is `/etc/bootptab`, and its syntax is detailed in the bootptab manpage. Let's create it.

First, open as root your favourite text editor. It is vim. Yes, it is. If it is not, it has to become. Now, enter the following lines (they are the default attributes). All the attributes you give here and do not override in a machine's specific attributes list will be given to clients):

```
        .default\
                :sm=your subnet mask\
                :ds=the IP address of your DNS server\
                :ht=ethernet\
                :dn=your domain name\
                :gw=the IP address of your gateway\
                :sa=the IP address of the TFTP server\
                :bf=path to find the kernel image\
                :rp=path of the root filesystem\
                :hn
```

Of course, not all these parameters are required, this depends on your network configuration and BOOTP implementations, but these will work in most cases.

Then, add an entry per client in your network. An entry should look like this:

```
        dns of the client\
                :ha=MAC address of the client\
                :ip=IP address of the client
```

The MAC address above is the hexadecimal hardware address of the client without the ':' characters.

Here is a sample `/etc/bootptab` file:

```
              .default\
                      :sm=255.255.0.0\
                      :ds=192.168.0.2\
                      :ht=ethernet\
                      :dn=frtest.org\
                      :gw=192.168.0.1\
```

```
                    :sa=192.168.0.2\
                    :bf=/tftpboot/kernel\
                    :rp=/nfsroot\
                    :hn

            foo\
                    :ha=001122334455\
                    :ip=192.168.2.12

            bar\
                    :ha=00FFEEDDCCBB\
                    :ip=192.168.12.42\
                    :ds=192.168.2.42
```

Then, run the bootpd daemon with the bootpd −s command (it is also a good idea to add it to your startup scripts), or add the following line to your `/etc/inetd.conf`:

```
        bootps dgram udp wait root /usr/sbin/tcpd bootpd −i −t 120
```

If you want to test the BOOTP server, add an entry to your `/etc/bootptab` and use the bootptest program.

4.3. TFTP

Setting up the TFTP daemon is not the hard part: just install the tftpd package if you have one, and add the following line to your `/etc/inetd.conf` (again, I assume `/tftpboot` is the directory where the kernel image resides):

```
        tftp dgram udp wait root /usr/sbin/tcpd in.tftpd /tftpboot
```

Don't forget to chmod 555 the `/tftpboot` directory, as most TFTP servers won't send the files if they are not world readable.

You should be aware of the limitations implied by running the TFTP daemon from the inetd. Most inetd's will hsutdown a service if it is spawned to frequently. So if you have many clients, you should look for another inetd like xinetd, or run a standalone TFTP daemon.

Now you have properly setup all daemons, you can restart the inetd and take a coffee. Don't forget to tell everyone the server setup is over, so you think you're a hero before you start building the root filesystem for the clients.

5. Clients setup, creation of the root filesystem

Tired? No you're not. Remember you're a hero. Here comes the tricky part. We will (err... *you* will) build the client's root filesystem. This shouldn't be very hard, but you probably will have to use trial and error.

The simplest way to create a root filesystem is to use an already working filesystem and customize it for the needs of diskless operation. Of course, you can also build one by hand (like in the good old times) if you like:=), but I won't explain this here.

5.1. Creating the first files and directories

First, **cd** to your future station's root directory. You can safely create the future `/home` directory with the **mkdir**, or by copying it from anywhere you want (you can use **cp –a** to do a recursive copy preserving owners, groups, symlinks, and permissions). Same thing for the future `/mnt`, `/root`, `/tmp` (don't forget to **chmod 0** it, this is only a mount point for the actual `/tmp` we will use, because each workstation needs to have its own `/tmp`). Then, copy some existing `/bin`, `/sbin`, `/boot`, and `/usr` into this future root directory (use **cp –a**). You can create the `/proc` directory with mkdir, and **chmod 0** it. Note some applications need write access to their user's home directory.

The `/lib` directory can be safely copied from somewhere else, but you will have to put the proper modules in it. To do so, use the following commands (assuming you have compiled the kernel for your clients on the server in `/usr/src/linux`, and the root filesystem will reside in `/nfsroot`):

```
# cd /usr/src/linux
        # make modules_install INSTALL_MOD_PATH=/nfsroot
```

Don't forget to put the `System.map` file in `/nfsroot/boot`. A first problem we will have to fix is that, depending on your configuration, your system may try to run fsck on the root filesystem at boot time. It shouldn't if there is no hard drive in the box. Most distributions will also skip this fsck if they find a `fastboot` file in the root directory. So, issue the following commands if you do not plan to mount any hard drive:

```
# cd /nfsroot
        # touch fastboot
        # chmod 0 fastboot
```

Another method is tell **fsck** that checking a NFS filesystem always succeeds:

```
# cd /nfsroot/sbin
      # ln –s ../bin/true fsck.nfs
```

The `/dev` directory can also be safely copied from another place into `/nfsroot`. But permissions and symlinks have to be preserved, so use **cp –a**. Another solution is to use kernel 2.2.x devfs feature, which will reduce memory consumption and improve performance, but the drawback of this method is that all symlinks created in `/dev` will be lost. The point to remember is that each workstation needs to have its own `/dev`, so you will have to copy it on a ramdisk if you plan to use several clients and not to use devfs.

5.2. The `/var` and `/etc` directories

We will use ramdisks for these directories, because each client needs to have its own one. But we still need them at the beginning to create their standard structure. Note you are not required to do so if you use a single client. So copy these directories (**cp –a**) from another place into `/nfsroot`. Then you can make some cleanup in `/var`: you can remove everything in `/nfsroot/var/log` and `/nfsroot/var/run`. You also probably can remove everything in `/nfsroot/var/spool/mail`, if you plan to export it via NFS. You also will have to remove the files containing host specific information in /nfsroot/etc to build them on the fly during the boot process.

The startup scripts will have to be customized in order to mount some parts of the filesystem: the `/dev` directory, if you don't use devfs, the `/tmp`, the `/var`, and the `/etc` directories. Here is some code which will achieve this:

```
        # this part only if you don't use devfs
        mke2fs -q -i 1024 /dev/ram0 16384
        mount -n -t ext2 -o rw,suid,dev,exec, \
            async,nocheck /dev/ram0 /dev
        # this part for everyone
        mke2fs -q -i 1024 /dev/ram1 16384
        mount -n -t ext2 -o rw,suid,dev,exec, \
            async,nocheck /dev/ram1 /tmp
        chmod 1777 /tmp
        cp -a /etc /tmp
        mke2fs -q -i 1024 /dev/ram2 16384
        mount -n -t ext2 -o rw,suid,dev,exec, \
            async,nocheck /dev/ram2 /etc
        find /tmp/etc -maxdepth 1 -exec cp -a '{}' /etc ';'
        mount -f -t ext2 -o rw,suid,dev,exec, \
            async,nocheck,remount /dev/ram2 /etc
        mount -f -o remount /
        cp -a /var /tmp
        mke2fs -q -i 1024 /dev/ram3 16384
        mount -t ext2 -o rw,suid,dev,exec, \
            async,nocheck /dev/ram3 /var
        find /tmp/var -maxdepth 1 -exec cp -a '{}' /var ';'
```

If you plan to use more than a single client, you will also have to change files dynamically at boot time in `/etc`: the files which contain the IP and hostname of the client. These files depend on your distribution, but you will easily find them with a few greps. Just remove client–specific information from them, and add code into your startup files to generate this information again at boot time *but only once the new `/etc` has been mounted on the ramdisk!* A way to obtain your IP address and hostname at bootup is the following (if you have the bootpc package installed on the workstations'filesystem):

```
        IPADDR="$(bootpc | awk '/IPADDR/ \
                                {
                                  match($0,"[A-Za-z]+")
                                  s=substr($0,RSTART+RLENGTH)
                                  match(s,"[0-9.]+")
                                  print substr(s,RSTART,RLENGTH)
                                }
                               ')"

        HOST="$(bootpc | awk '/HOSTNAME/ \
                              {
```

```
                                match($0,"[A-Za-z]+")
                                s=substr($0,RSTART+RLENGTH)
                                match(s,"[A-Za-z0-9-]+")
                                print substr(s,RSTART,RLENGTH)
                              }')"

        DOMAIN="$(bootpc | awk '/DOMAIN/ \
                                {
                                  match($0,"[A-Za-z]+")
                                  s=substr($0,RSTART+RLENGTH)
                                  match(s,"[A-Za-z0-9-.]+")
                                  print substr(s,RSTART,RLENGTH)
                                }')"
```

This is a complicated solution, but I guess it should work on most sites. The IP address can alternatively be obtained with the output of **ifconfig** and the hostname can be obtained from the output of the host command, but this is not portable, because these outputs differ from system to system depending on the distribution you are using, and the locales settings.

Then, the hostname should be set with the **hostname $HOSTNAME** command. When this is done, it is time to generate on the fly the configuration files which contain the IP address or the hostname of the client.

5.3. Last details

Now, it's time to do the fine tuning of the client. As `/var` will be mounted on a ramdisk (unless you have a single client), you will have to send the logs to a logs server if you want to keep them. One way to do that is to delete the `/nfsroot/etc/syslog.conf` file and replacing it by the following file (see man syslog.conf for details):

```
        *.*     /dev/tty12
        *.*     @dns or IP of the logs server
```

If you do so, the logs server will have to run **syslogd** with the –r option (see the syslogd manual page).

If you use **logrotate** and you have done the preceding operation, you should replace the **logrotate** configuration file (`/etc/logrotate.conf` on most boxes) by an empty file:

```
# rm -f /etc/logrotate.conf
        # touch /etc/logrotate.conf
```

If you don't use it, just remove the logs rotation scripts from the crontab, and as you no longer have log files in `/var/log`, put an **exit 0** at the beginning of your logs rotation scripts.

In the `/nfsroot/etc/fstab` file, remove anything related to the hard drive, floppy disk reader, or cdrom if you don't have such devices on your workstations. Add an entry for the `/var/spool/mail` directory, which should be exported by the server through NFS or any other network filesystem. You probably also want to put an entry for the `/home` directory in this file.

You can also comment the lines running newaliases, activating swap, and running depmod –a and remove the `/nfsroot/etc/mtab` file. Comment out the line(s) removing `/fastboot`, `/fsckoptions`, and `/forcefsck` in your startup scripts. Also remove or comment any line in the startup scripts that would try to write on the root filesystem except for really necessary writes, which should all be redirected to some ramdisk location if you use several clients.

5.4. Trial...

Time has come for a small trial. MAKE A BACKUP OF YOUR NEWLY CREATED `/nfsroot`. tar −cvvIf should do fine. Take a minute to verify we didn't forget anything. Try to boot a client.

5.5. And Error!

Look carefully at the client's screen during the boot process. Oh, I didn't tell you to connect a screen... Run, forest! Run an get one. You will probably see some error messages. Fix the problems, and make frequent backups of your `/nfsroot`. One day, the client will boot properly. This day, you will have to fix errors occurring during shutdown;=P.

6. Several ways of obtaining the bootloader's code

We have spoken so far about the client and server's configuration for operation after the BOOTP request has been issued by the client, but the first problem is that most computers are not able to behave as BOOTP clients by default. We will see in this section how to fix this.

6.1. BOOTP or DHCP capable NICs

This is the most simple case: some network cards provide a supplement to the BIOS, containing a BOOTP or DHCP client, so just setup them for BOOTP or DHCP operation in the BIOS, and you're done.

6.2. Local floppy or hard drive

These cases are also quite simple: the kernel is loaded from a local drive, and all the kernel has to do is to obtain its network parameters from BOOTP, and mount its root filesystem over NFS; this should not cause any problem. By the way, a local hard drive is a good place to leave a `/var`, `/tmp`, and a `/dev`...

If you have a local hard drive, all you have to do is to use lilo or your favourite boot loader as usual. If you use a floppy, you can use a bootloader or simply write the kernel on the floppy: a kernel is directly bootable.This enables you to use a command like the following:

```
# dd if=zImage of=/dev/fd0 bs=8192
```

6.3. Creating ROMs for the clients

Many network cards include a slot in which one can insert an EPROM with additional BIOS code. This enables one to add, for instance, BOOTP capabilities to the BI0S. To do so, you will first have to find how to enable the EPROM socket. You may need a jumper or a special software to do so. Some cards like the 3Com 905B have slots for EEPROMs which enable one to change the software in the EEPROM in place. In appendix, you'll find the information about EPROM and various types of memory chips.

For a list of EPROM burner manufacturers visit the Yahoo site and go to economy−>company−>Hardware−>Peripherals−>Device programmers or check out the old Diskless−HOWTO *List of EPROM burner manufacturers* section.

If you choose to create your own ROMS, you will have to load a BOOTP or DHCP capable software in the ROM, and then, you will be in the case of BOOTP or DHCP capable NICs described above.

You will also need to find the proper EPROM size and speed for your NIC. Some methods to do so are provided in appendix, because the NICs manufacturers often do not provide this information.

6.3.1. LanWorks BootWare PROMs

This information may save you time. In order to make LanWorks BootWare(tm) PROMs to correctly start up a linux kernel image, the "bootsector" part of the image must be modified so as to enable the boot prom to jump right into the image start address. The net−bootable image format created by netboot/etherboot's `mknbi−linux' tool differs and will not run if used with BootWare PROMs.

A modified bootsector together with a Makefile to create a BootWare−bootable image after kernel compilation can be found at:

- Bwimage package: ftp://ftp.ipp.mpg.de/pub/ipp/wls/linux/bwimage−0.1.tgz
- See also http://www.patoche.org/LTT/net/00000096.html
- LanWorks BootWare Boot ROMs: http://www.3com.com/lanworks

Refer to the README file for installation details. Currently, only "zImage"−type kernels are supported. Unfortunately, kernel parameters are ignored.

This section was initially written by Jochen Kmietsch for the Diskless−HOWTO, email to: `<jochen.kmietsch@tu-clausthal.de>` for any questions.

6.4. Local CDROM

This section was originally written by Hans de Goede `<j.w.r.degoede@et.tudelft.nl>` for the Diskless−root−NFS−HOWTO. I modified it slightly in order to reflect some differences between this document and the Diskless−root−NFS−HOWTO.

Much of the above also goes for booting from cdrom. Why would one want to boot a machine from cdrom? Booting from cdrom is interesting everywhere one wants to run a very specific application, like a kiosk, a library database program or an internet cafe, and one doesn't have a network or a server to use a root over nfs setup.

6.4.1. Creating a test setup

Now that we know what we want to do and how, it's time to create a test setup:

- For starters just take one of the machines which you want to use and put in a big disk and a cd burner.
- Install your linux of choice on this machine, and leave a 650 MB partition free for the test setup. This install will be used to make the iso image and to burn the cd's from, so install the necessary tools. It will also be used to restore any booboo's which leave the test setup unbootable.
- On the 650 mb partition install your linux of choice with the setup you want to have on the cd, this will be the test setup.
- Boot the test setup.
- Compile a kernel with isofs and cdrom support compiled in.
- Configure the test setup as described above with the root filesystem mounted read only.
- Verify that the test setup automagically boots and everything works.
- Boot the main install and mount the 650 MB partition on `/test` of the main install.
- Put the following in a file called `/test/etc/rc.d/rc.iso`, this file will be sourced at the

beginning of `rc.sysinit` to create `/var`:

```
#/var
echo Creating /var ...
mke2fs -q -i 1024 /dev/ram1 16384
mount /dev/ram1 /var -o defaults,rw
cp -a /lib/var /
```

- Edit `/test/etc/rc.sysinit`, comment the lines where the root is remounted rw, and add the following 2 lines directly after setting the PATH:

```
#to boot from cdrom
. /etc/rc.d/rc.iso
```

- Copy the following to a script and execute it to make a template for `/var` and create `/tmp` and `/etc/mtab` links.

```
#!/bin/sh
echo tmp
rm -fR /test/tmp
ln -s var/tmp /test/tmp

###
echo mtab
touch /test/proc/mounts
rm /test/etc/mtab
ln -s /proc/mounts /test/etc/mtab

###
echo var
mv /test/var/lib /test/lib/var-lib
mv /test/var /test/lib
mkdir /test/var
ln -s /lib/var-lib /test/lib/var/lib
rm -fR /test/lib/var/catman
rm -fR /test/lib/var/log/httpd
rm -f /test/lib/var/log/samba/*
for i in `find /test/lib/var/log -type f`; do
  cat /dev/null > $i;
done
rm `find /test/lib/var/lock -type f`
rm `find /test/lib/var/run -type f`
```

- Remove the creation of `/etc/issue*` from `/test/etc/rc.local`: it will only fail.
- Now boot the test partition again, it will be read only just like a cdrom. If something doesn't work reboot to the working partition fix it, try again etc. Or you could remount / rw, fix it, then reboot straight into to test partition again. To remount / rw type:

```
# mount -o remount,rw /
```

6.4.2. Creating the CD

If you need more information than you can find below, please refer to the CD−Writing−HOWTO.

6.4.2.1. Creating a boot image

First of all, boot into the working partition. To create a bootable cd we'll need an image of a bootable floppy. Just dd–ing a zImage doesn't work since the loader at the beginning of the zimage doesn't seem to like the fake floppydrive a bootable cd creates. So we'll use syslinux instead.

- Get boot.img from a redhat cd.
- Mount boot.img somewhere through loopback by typing:

```
# mount boot.img somewhere -o loop -t vfat
```

- Remove everything from boot.img except for ldlinux.sys and syslinux.cfg.
- Cp the kernel–image from the test partition to boot.img.
- Edit `syslinux.cfg` so that it contains the following, of course replace `zImage` by the appropriate image name:

```
                default linux

                label linux
                kernel zImage
                append root=/dev/<insert your cdrom device here>
```

- Umount boot.img:

```
# umount somewhere
```

- If your `/etc/mtab` is a link to `/proc/mounts`, umount won't automagically free `/dev/loop0` so free it by typing:

```
# losetup -d /dev/loop0
```

6.4.2.2. Creating the iso image

Now that we have the boot image and an install that can boot from a readonly mount it's time to create an iso image of the cd:

- Copy boot.img to `/test`
- Cd to the directory where you want to store the image and make sure it's on a partition with enough free space.
- Now generate the image by typing:

```
# mkisofs -R -b boot.img -c boot.catalog -o boot.iso /test
```

6.4.2.3. Verifying the iso image

- Mounting the image through the loopbackdevice by typing:

```
# mount boot.iso somewhere -o loop -t iso9660
```

- Umount boot.iso:

```
# umount somewhere
```

- If your `/etc/mtab` is a link to `/proc/mounts` umount won't automagically free `/dev/loop0` so free it by typing:

```
# losetup -d /dev/loop0
```

6.4.2.4. Writing the actual CD

Assuming that you've got **cdrecord** installed and configured for your cd−writer type:

```
# cdrecord -v speed=<desired writing speed> dev=<path to your writers generic scsi device> boot.iso
```

6.4.3. Boot the cd and test it

Well the title of this paragraph says it all;)

7. How to create diskless MS−Windows stations?

Since MS−Windows does not support diskless booting, a simple workaround is presented here: the solution is to use software like VMWare or its free alternative, FreeMWare. These enable MS−Windows to be executed transparently on the linux box.

8. Troubleshooting, tips, tricks, and useful links

8.1. The netboot distribution

This distribution contains a bootloader enabling one to boot diskless stations with either Linux or DOS. This is an alternative to the usage of etherboot.

8.2. Reducing diskless workstations'memory usage

One simple way to reduce memory consumption is to put several dynamically created directories on the same ramdisk. For instance, let's say the first ramdisk will contain the `/tmp` directory. Then, one may move the `/var/tmp` directory on that ramdisk with the following commands issued on the server:

```
# mkdir /nfsroot/tmp/var
        # chmod 0 /nfsroot/tmp/var
        # ln -s /tmp/var /nfsroot/var/tmp
```

Another good way to reduce memory consumption if you don't have local hard drives and do not swap over a network block device is to disable the **Swapping to block devices** option during kernel compilation.

8.3. Swapping over NFS

If your stations do not have enough memory and do not have local drives, you may want to swap over NFS. You have to be warned the cod eto do so is still under developpement and this method is generally quite slow. The full documentation for this can be found at http://www.instmath.rwth−aachen.de/~heine/nfs−swap/.

The first thing to do if you want to apply this solution is to patch your kernel (you need a kernel version 2.2 or above). First download the patch at the above url, and cd to `/usr/src/linux`. I assume the patch is in `/usr/src/patch`. Then issue the following command:

```
# cat ../patch | patch -p1 -l -s
```

Then, compile your kernel normally and enable the **Swapping via network sockets (EXPERIMENTAL)** and **Swapping via NFS (EXPERIMENTAL)** options.

Then export a directory read−write and no_root_squash from the NFS server. Setup the clients so that they will mount it somewhere (say on `/mnt/swap`). It should be mounted with a rsize and wsize smaller than the page size used by the kernel (ie. 4 kilobytes on Intel architectures), otherwise your machine may run out of memory due to memory fragmentation; see the nfs manual page for details about rsize and wsize. Now, to create a 20 MB swap file, issue the following commands (which should be placed in the clients'initialization scripts):

```
# dd if=/dev/zero of=/mnt/swap/swapfile bs=1k count=20480
# mkswap /mnt/swap/swapfile
# swapon /mnt/swap/swapfile
```

Of course, this was just for an example, because if you have several workstations, you will have to change the swap file name or directory, or all your workstations will use the same swap file for their swap...

Let's say a word about the drawbacks of NFS swapping: the first drawback is that it is generally slow, except you have specially fast network cards. Then, this possibility has not been very well tested yet. At last, this is not secure at all: anyone on the network is able to read the swapped data.

8.4. Swapping over network block devices

I have to warn you I never tryed this, and I would particularly appreciate feedback about this section.

The general principle for swapping over network block devices is the same than to swap over NFS. The good point is you won't have to patch the kernel. But most of the same drawbacks also apply to the NBD method.

To create a 20 MB swap file, you will have to first create it on the server, export it to the client, and do an **mkswap** on the file. Note that the **mkswap** must be done on the server, because mkswap uses system calls which are not handled by NBD. Moreover, this command must be issued after the server starts exporting the file, because the data on the file may be destroyed when the server starts exporting it. If we assume the server's name is NBDserver, the client's name is NBDclient, and the TCP port used for the export is 1024, the commands to issue on the server are the following:

```
# dd if=/dev/zero of=/swap/swapfile bs=1k count=20480
# nbd-server NBDclient 1024 /swap/swapfile
# mkswap /swap/swapfile
```

Now, the client should use the following command:

```
# swapon /dev/nd0
```

Again, this was just to show the general principle. The files'names should also be dependant on the workstations'names or IPs.

Another solution to swap over a network block device is to create an ext2 filesystem on the NBD, then create a regular file on this filesystem, and at last, use **mkswap** and **swapon** to start swapping on this file. This second method method is closer to the swap over NFS method than the first solution.

8.5. Getting rid of error messages about `/etc/mtab` or unmounted directories on shutdown

The following commands, issued on the server may solve the problem:

```
# ln -s /proc/mounts /nfsroot/etc/mtab
        # touch /nfsroot/proc/mounts
```

8.6. Installing new packages on workstations

A simple way to do so is to use, on the server, a chroot and then execute your favourite installation commands normally. To chroot to the appropriate place, use the following command:

```
# chroot /nfsroot
```

Debian users will be particularly interested in the −−root option of dpkg, which simply tells dpkg where the root of the target system is.

A. Non−Volatile Memory chips

Here is a brief descriptions of memory chips and their types:

- PROM: Pronounced prom, an acronym for programmable read−only memory. A PROM is a memory chip on which data can be written only once. Once a program has been written onto a PROM, it remains there forever. Unlike RAM, PROMs retain their contents when the computer is turned off. The difference between a PROM and a ROM (read−only memory) is that a PROM is manufactured as blank memory, whereas a ROM is programmed during the manufacturing process. To write data onto a PROM chip, you need a special device called a PROM programmer or PROM burner. The process of programming a PROM is sometimes called burning the PROM. An EPROM (erasable programmable read−only memory) is a special type of PROM that can be erased by exposing it to ultraviolet light. Once it is erased, it can be reprogrammed. An EEPROM is similar to a PROM, but requires only electricity to be erased.
- EPROM: Acronym for erasable programmable read−only memory, and pronounced e−prom, EPROM is a special type of memory that retains its contents until it is exposed to ultraviolet light. The ultraviolet light clears its contents, making it possible to reprogram the memory. To write to and erase an EPROM, you need a special device called a PROM programmer or PROM burner. An EPROM differs from a PROM in that a PROM can be written to only once and cannot be erased. EPROMs are used widely in personal computers because they enable the manufacturer to change the contents of the PROM before the computer is actually shipped. This means that bugs can be removed and new versions installed shortly before delivery. A note on EPROM technology: The bits of an EPROM are programmed by injecting electrons with an elevated voltage into the floating gate of a field−effect transistor where a 0 bit is desired. The electrons trapped there cause that transistor to conduct, reading as 0. To erase the EPROM, the trapped electrons are given enough energy to escape the floating gate by bombarding the chip with ultraviolet radiation through the quartz window. To prevent slow erasure over a period of years from sunlight and fluorescent lights, this quartz window is covered with an opaque label in normal use.
- EEPROM: Acronym for electrically erasable programmable read−only memory. Pronounced double−e−prom or e−e−prom, an EEPROM is a special type of PROM that can be erased by exposing it to an electrical charge. Like other types of PROM, EEPROM retains its contents even when the power is turned off. Also like other types of ROM, EEPROM is not as fast as RAM. EEPROM is similar to flash memory (sometimes called flash EEPROM). The principal difference is that EEPROM requires data to be written or erased one byte at a time whereas flash memory allows data to be written or erased in blocks. This makes flash memory faster.
- FRAM: Short for Ferroelectric Random Access Memory, a type of non−volatile memory developed by Ramtron International Corporation. FRAM combines the access speed of DRAM and SRAM with the non−volatility of ROM. Because of its high speed, it is replacing EEPROM in many devices. The term FRAM itself is a trademark of Ramtron.
- NVRAM: Abbreviation of Non−Volatile Random Access Memory, a type of memory that retains its contents when power is turned off. One type of NVRAM is SRAM that is made non−volatile by connecting it to a constant power source such as a battery. Another type of NVRAM uses EEPROM chips to save its contents when power is turned off. In this case, NVRAM is composed of a combination of SRAM and EEPROM chips.
- Bubble Memory: A type of non−volatile memory composed of a thin layer of material that can be easily magnetized in only one direction. When a magnetic field is applied to circular area of this substance that is not magnetized in the same direction, the area is reduced to a smaller circle, or bubble. It was once widely believed that bubble memory would become one of the leading memory technologies, but these promises have not been fulfilled. Other non−volatile memory types, such as EEPROM, are both faster and less expensive than bubble memory.

- Flash Memory: A special type of EEPROM that can be erased and reprogrammed in blocks instead of one byte at a time. Many modern PCs have their BIOS stored on a flash memory chip so that it can easily be updated if necessary. Such a BIOS is sometimes called a flash BIOS. Flash memory is also popular in modems because it enables the modem manufacturer to support new protocols as they become standardized.

B. Determining the size and speed of EPROMs to plug in a NIC

This section comes from the etherboot project's documentation version 5.0. It provides tips to determine the size and speed of EPROMs to use with a particular NIC

The smallest EPROM that is accepted by network cards is an 8k EPROM (2764). 16 kB (27128) or 32 kB (27256) are the norm. Some cards will even go up to 64 kB EPROMs (27512). (You will often see a C after the 27, e.g. 27C256. This indicates a CMOS EPROM, which is equivalent to the non−C version and is a good thing because of lower power consumption.) You want to use the smallest EPROM you can so that you don't take up more of the upper memory area than needed as other extensions BIOSes may need the space. However you also want to get a good price for the EPROM. Currently the 32 kB and 64 kB EPROMs (27256 and 27512) seem to be the cheapest per unit. Smaller EPROMs appear to be more expensive because they are out of mainstream production.

If you cannot find out from the documentation what capacity of EPROM your card takes, for ISA NICs only, you could do it by trial and error. (PCI NICs do not enable the EPROM until the BIOS tells the NIC to.) Take a ROM with some data on it (say a character generator ROM) and plug it into the socket. Be careful not to use an extension BIOS for this test because it may be detected and activated and prevent you from booting your computer. Using the debug program under DOS, dump various regions of the memory space. Say you discover that you can see the data in a memory window from CC00:0 to CC00:3FFF (= 4000 hex = 16384 decimal locations). This indicates that a 16 kB EPROM is needed. However if you see an alias in parts of the memory space, say the region from CC00:0 to CC00:1FFF is duplicated in CC00:2000 to CC00:3FFF, then you have put an 8 kB EPROM into a 16 kB slot and you need to try a larger EPROM.

Note that because pinouts for 28 pin EPROMs are upward compatible after a fashion, you can probably use a larger capacity EPROM in a slot intended for a smaller one. The higher address lines will probably be held high so you will need to program the image in the upper half or upper quarter of the larger EPROM, as the case may be. However you should double check the voltages on the pins armed with data sheet and a meter because CMOS EPROMs don't like floating pins.

If the ROM is larger than the size of the image, for example, a 32 kB ROM containing a 16 kB image, then you can put the image in either half of the ROM. You will sometimes see advice to put two copies of the image in the ROM. This will work but is not recommended because the ROM will be activated twice if it's a legacy ROM and may not work at all if it's a PCI/PnP ROM. It is tolerated by Etherboot because the code checks to see if it's been activated already and the second activation will do nothing. The recommended method is to fill the unused half with blank data. All ones data is recommended because it is the natural state of the EPROM and involves less work for the PROM programmer. Here is a Unix command line that will generate 16384 bytes of 0xFF and combine it with a 16 kB ROM into a 32 kB image for your PROM programmer.

```
# (perl −e 'print "\xFF" x 16384'; cat bin32/3c509.lzrom) > 32kbimage
```

The speed of the EPROM needed depends on how it is connected to the computer bus. If the EPROM is directly connected to the computer bus, as in the case of many cheap NE2000 clones, then you will probably have to get an EPROM that is at least as fast as the ROMs used for the main BIOS. This is typically 120−150 ns. Some network cards mediate access to the EPROM via circuitry and this may insert wait states so that slower EPROMs can be used. Incidentally the slowness of the EPROM doesn't affect Etherboot execution speed much because Etherboot copies itself to RAM before executing. I'm told Netboot does the same thing.

If you have your own EPROM programming hardware, there is a nice collection of EPROM file format conversion utilities at http://www.canb.auug.org.au/~millerp/srecord.html. The files produced by the Etherboot build process are plain binary. A simple binary to Intel hex format converter can be found at the Etherboot web site at http://etherboot.sourceforge.net/bin2intelhex.c.

Etherboot is believed to make PnP compliant ROMs for PCI NICs. A long−standing bug in the headers has been tracked down. However some faulty old BIOSes are out there so I have written a Perl script swapdevids.pl to switch the header around if necessary. You'll have to experiment with it both ways to find out which works. Or you could dump a ROM image that works (e.g. RPL, PXE ROM) using the Perl script disrom.pl. The fields to look at are Device (base, sub, interface) Type. It should be 02 00 00, but some BIOSes want 00 00 02 due to ambiguity in the original specification.

C. Companies selling diskless computers

The original Diskless−HOWTO mentions the names of the following vendors of diskless computers:

- Linux Systems Labs Inc., USA http://www.lsl.com. Click on "Shop On−line" and then click on "HardWare" where all the diskless computers will be listed. Phone 1−888−LINUX−88.
- Diskless Workstations Corporation, USA http://www.disklessworkstations.com.
- Unique Systems of Holland Inc., Ohio, USA http://www.uniqsys.com

References

[Diskless−HOWTO] *Diskless−HOWTO*.

http://www.linuxdoc.org/HOWTO/Diskless−HOWTO.html

[Diskless−root−NFS−HOWTO] *Diskless−root−NFS−HOWTO*.

http://www.linuxdoc.org/HOWTO/Diskless−root−NFS−HOWTO.html

[ltsp] *Linux Terminal Server Project*.

http://www.ltsp.org

[logilab] *Logilab.org web site*.

http://www.logilab.org

[PowerUp2Bash] *From−PowerUp−to−bash−prompt−HOWTO*.

http://www.linuxdoc.org/HOWTO/From−PowerUp−to−bash−prompt−HOWTO.html

[ThinClient] *Thin−Client−HOWTO*.

http://www.linuxdoc.org/HOWTO/Thin−Client−HOWTO.html

[cdwriting] *CD−Writing−HOWTO*.

http://www.linuxdoc.org/HOWTO/CD−Writing−HOWTO.html

[etb] *etherboot project*.

http://etherboot.sourceforge.net/

[vmware] *VMWare*.

http://www.vmware.com

[freemware] *FreeMWare*.

http://www.freemware.org