Linux Boot Loaders Compared

L.C. Benschop

May 29, 2003

Contents

1 introduction

If you use Linux on a production system, you will only see it a few times a year. If you are a hobbyist who compiles many kernels or who uses many operating systems, you may see it several times per day. Of course I mean the boot loader.

There are several different ways to boot Linux and every method has specific advantages and drawbacks. Therefore it is important to know which boot loading methods exist for Linux and how they compare to each other.

We discuss only boot loaders that meet the following criteria:

- They must support Linux. The boot loader must load the Linux kernel itself, boot loaders that could chain load another loader, which in turn boots Linux, are not discussed,
- They must run on the Intel PC platform.
- They must be freely available complete with source code.
- No network boot loaders are discussed.

We discuss the following boot loaders:

- Boot Sector of the Linux kernel.
- LILO
- GNU GRUB
- SYSLINUX

- LOADLIN

Other less well-known boot loaders do exist and may support the criteria. I know of the following:

- nuni[1] is a boot loader that does not use BIOS, but is otherwise rather limited. It boots *only* from IDE disks.
- Gujin[2] is a boot loader that understands file systems, just like GRUB.
- `mbr03` and `e2boot`[3] are an MBR boot selector and a matching Linux boot loader that fits into the first 1k boot block of an ext2 partition, so no file space is used by them.

Note: some boot loaders are rapidly evolving, so some features may have been added after this article was written.

2 How Boot Loaders Work

2.1 What BIOS does for us

The BIOS is the firmware in the ROM of a PC. When the PC is powered up, the BIOS is the first program that runs. All other programs must be loaded into RAM first. The BIOS contains the following parts:

- POST (Power On Self Test). The running counter that counts the kilobytes of main memory is the most visible part of the POST.
- The Setup Menu, that lets you set some parameters and lets you adjust the real time clock. Most modern BIOS versions let you set the boot order, the devices that BIOS checks for booting. These can be A (the first floppy disk), C (the first hard disk), CD-ROM and possibly other disks as well. The first device in the list will be tried first. Older BIOS-es have only one boot order: A, C. So the BIOS will try to boot from A first and if there is no diskette in the drive it tries to boot from C.
- The boot sector loader. This loads the first 512-byte sector from the boot disk into RAM and jumps to it. This is where the boot loaders described in this paper start.

[1] http://www.ibiblio.org/pub/Linux/system/boot/loaders/
[2] http://gujin.sourceforge.net
[3] ftp://spruce.he.net/pub/jreiser/

- The BIOS interrupts. These are simple device drivers that programs can use to access the screen, the keyboard and disks. Boot loaders rely on them, most operating systems do not (the Linux kernel does not use BIOS interrupts once it has been started). MSDOS does use BIOS interrupts.

Apart from the main BIOS there are extension ROMs, which are started by the main BIOS. Every VGA card has one. Also SCSI host adapters and Ethernet cards can have an extension ROM. It is even possible to put an EPROM on an Ethernet card to boot a machine over the network without any disks.

As far as boot loading facilities are concerned, the PC BIOS is very primitive compared to that of other computer systems. The only thing it knows about disks is how to load the first 512-byte sector.

- The first sector of a diskette can be loaded at address 0000:7C00. The last two bytes of the sector are checked for the values 0x55 and 0xAA, this as a rough sanity check. If these are OK, the BIOS jumps to the address 0000:7C00.

- Booting from a hard disk is very similar to booting from a diskette. The first sector of a hard disk (often called the Master Boot Record) is loaded at 0000:7C00 and next the BIOS jumps to it. The MBR program must move itself to an address that is different from 0000:7C00 as it is supposed to load a different boot sector from a partition to address 0000:7C00 and jump to that.

- Modern BIOS versions can treat a certain file on a CD-ROM as a diskette image. They pretend to boot from a diskette by loading the first 512 bytes of the file to 0000:7C00 and jumping to it. Every attempt to access the same diskette using the BIOS routines, will be redirected to the image file on CD-ROM. Some other ways to boot a CD-ROM may also be supported (with an emulated hard disk or with no disk emulation at all).

When the boot sector is loaded, the CPU is in real mode. For those who are unfamiliar with 80x86 architecture: real mode is very limited compared to 32-bit protected mode (in which Linux runs). For example: data outside a 64K segment can only be accessed if you change a segment register and data outside the first 1MB of address space (which contains 640kB of main memory) cannot be accessed at all. As `gcc` does not know about real mode, programs compiled with it can only be run in real mode with some tricks and with severe memory size restrictions. This is the reason why most boot loaders (except GRUB) are written in assembly. All boot sector programs, even that of GRUB, are written in

assembly[4].

In theory a boot loader could do its job by directly accessing the bare metal, but 512 bytes would be a very small space for that. The boot loader has access to BIOS interrupts, which are subroutines that can be invoked by the INT instruction (software interrupts). These work in real mode only. The following routines are used by most boot loaders.

- INT 0x10 for screen output.
- INT 0x16 for keyboard input.
- INT 0x13 for disk I/O. The parameters to specify sectors on disk used to have a very limited range. Originally it was only possible to specify 1024 cylinders on a hard disk, while hard disks can have more cylinders. This imposed limitations on where it was allowed to put the boot loader and any files accessed by it. You were forced to create a small partition near the start of the hard disk and put the boot loader there. There are three categories of BIOS:
  - BIOS versions earlier than 1995 could only access IDE disks of around 500MB, as the BIOS sector and head numbers corresponded directly to the register values on the IDE interface.
  - BIOS versions between 1995 and 1998 can access IDE disks up to about 8GB. They translate the cylinder, head and sector numbers from the INT 0x13 call to different values that better utilize the allowable ranges on the IDE interface.
  - BIOS versions of 1998 or later have a new calling interface using linear block addresses.

  In any of the three categories you have BIOS-es that have bugs that cause them to stop at a lower disk size limit (a category 2 BIOS that should work up to 8GB, but stops at 2GB). In those cases it makes sense to upgrade to a new BIOS version.
- INT 0x15 is a catch-all for many BIOS functions, one of which is moving data to extended memory (the BIOS is required to switch to protected mode temporarily to do that). Other functions are for disabling the A20 gate and for determining the memory size.

[4]Real mode was not one of the strongest points of the GNU assembler, not until recent versions. Therefore two other assemblers have been used for real-mode programs such as boot loaders. These are `as86` (in use by LILO and until recently by the real mode code in the Linux kernel) and `nasm` (in use by SYSLINUX).

2.2 Parts of a boot loader

A boot loader typically consists of three programs:

- The boot sector program is directly loaded by the BIOS at boot time and is only 512 bytes in size.
- The second stage program is loaded by the boot sector program and it does everything you expect the boot loader to do.
- The boot loader installer is not run when the system is booted, but it is used to install the boot loader and the second stage program onto the boot disk. These have to be stored in special locations, so they cannot be copied with `cp`.

2.2.1 boot sector program

The boot sector program can only be 512 bytes in size and not all 512 bytes are even available in all cases. The last two bytes must be 0x55 and 0xAA for the BIOS. The Master Boot Record on a hard disk contains the partition table of 64 bytes, so only the first 446 bytes can be used. If the boot sector program must exist on a DOS partition or on a DOS diskette, there must be a parameter block at the start of the boot sector.

Because of these size restrictions, boot sector programs are just about the only remaining examples of programs on the PC platform that are truly optimized for size and have to be written in assembly for this reason. Further, a boot sector program cannot do everything you want a boot loader to do. Usually a boot sector program does one of the following things (not all three in one program):

- Load another boot sector. This is typical for a boot sector program that lives in the master boot record of a hard disk. It can find the first sector of the selected *active* partition and *chain load* that. The MBR program that came traditionally with MS-DOS has no ability to change the active partition at boot time. There are other boot sector programs that let you select a partition by pressing a key, such as the MBR program of LILO.
- Load a second stage boot loader. It is generally not possible for a boot sector program to look into the directory for a file with a specific name and load that into memory, but exceptions exist, at least for DOS file systems[5]. Most boot sector programs find the second stage by sector number rather than by

[5]For an interesting example see the file `sector.S` in the `diagnose` subdirectory of the LILO boot loader. The `SYSLINUX` program does almost the same thing, but really loads just the first part of the second stage loader, which in turn loads the rest.

name. The sector numbers have to be put into the boot sector by the boot loader installer.

- Load the kernel directly. A kernel is typically much larger than a second stage boot loader.

  The boot sector program in the Linux kernel loads the kernel directly into memory without the need for a second stage boot loader. As the kernel is located in contiguous sectors on a diskette, there is no need to traverse file system data structures. However, for `bZimage` kernels the boot sector program cheats; it invokes a subroutine in the `setup` part of the kernel for loading the rest of the kernel into high memory.

  The boot loader `e2boot` fits into the first 1kB block of an ext2 partition (it is twice as big as a boot sector program), but with some tricks it finds both the kernel and the RAM disk by name on an ext2 partition and loads them into memory.

  Also the boot sector on a DOS disk does not utilize a second stage boot loader to load the MS-DOS kernel files `IO.SYS` and `MSDOS.SYS`. The structure of an MSDOS file system is simple enough to find a file with a specific name in the root directory and load it into memory, at least part of it..

2.2.2 second stage of boot loader

This is the real boot program. It contains the user interface and the kernel loader. It can be anywhere from 6.5 kilobytes (LILO) to over 100 kilobytes (GRUB) in size. It contains the following functions:

- User interface. It is either a simple command line (old versions of LILO), a menu or both. It allows you to select any number of operating systems and to specify additional parameters to the operating system. The available options are specified by a configuration file. Modern versions of boot loaders can show their menu in a bitmap picture.

- Operating system loader. loads the operating system into memory and runs it. Alternatively we can load another boot loader specific to another operating system and let it run. This is called chain loading.

`LOADLIN` is not a complete boot loader, but it has only the second stage (without the user interface). It is run from DOS, and it can make use of DOS system calls to read files from disk. What makes its task harder than that of a normal boot loader, it that it must be able to work its way out of some types of memory managers.

2.2.3 Boot loader installer

The third part of the boot loader is only run when the boot loader is installed on a disk. As opposed to the boot sector program and second stage, this is a normal Linux program. In the case of LILO the installer must be rerun each time the configuration is changed or any file has been updated. It performs the following tasks:

- Install the boot sector. If the boot sector will be installed in the MBR of a hard disk or on a DOS file system, not all 512 bytes may be overwritten, but the *partition table* or the *DOS parameter block* must be preserved.
- Tell the boot sector where the second stage boot loader is. Usually it writes one or more sector addresses into the boot loader.
- Tell the second stage boot loader where all relevant information is (configuration, kernels). This is the case with LILO. LILO creates a map file that contains all relevant sector addresses and puts pointers to the map file in the boot sector and/or second stage boot loader.

2.3 Loading the operating system

2.3.1 Loading the Linux kernel

A Linux kernel consists of the following parts:

- The boot sector (see `arch/i386/boot/bootsect.S`). This is only run if Linux is booted directly from a diskette.
- The setup part with real mode initialization code (transition to protected mode) (see `arch/i386/boot/setup.S`)
- The rest of the kernel, which in turn consists of the following parts:
  - Protected mode initialization code (see `arch/i386/boot/compressed/head.S`).
  - A decompression program (compiled C code, see `arch/i386/boot/compressed/misc.c` and `lib/inflate.c`).
  - The real kernel, which is compressed. After decompression this kernel consists of the following parts:
    * Protected mode initialization code (see `arch/i386/boot/head.S`).
    * Main C routine (see `init/main.c`).

    * All the rest. Relevant for this discussion is the RAM disk driver (see `drivers/block/rd.c`). This will be further explained in section 4

A Linux boot loader should support the following tasks:

- Loading the Linux kernel into memory.
  - The boot sector (which will not be run) and the setup part of the kernel are loaded near the top of low memory (usually address 9000:0000).
  - If it is not a `bzImage` type kernel, the rest of the kernel will be loaded in low memory at 0000:1000).
  - If it is a `bzImage` type kernel, the rest of the kernel will be loaded in high memory, starting at 0x100000.

  The boot loader does not know or care that the rest of the kernel contains a compressed part.
- Passing a command line to the kernel. This command line can come from a configuration file or it can be entered interactively by the user.
- Loading an initial RAM disk into memory and passing it to the kernel. The initial RAM disk will be loaded near the top of high memory above the kernel.
- Starting the kernel. The boot loader must set some relevant parameters in the setup part and jump to it. From there the kernel takes control and the boot loader's part is over. Next the kernel starts as follows:
  - The setup code saves relevant parameters and the command line for later.
  - The setup code checks how much memory is available, performs some hardware initialization and prepares to enter protected mode.
  - In protected code, the rest of the kernel is being decompressed[6].
  - After decompression the `head.S` and the main C routine will initialize all parts of the kernel. This will show a lot of messages.
  - Finally the `init` process will be started (or first its `linuxrc` variant on an initial RAM disk).

[6] At this point we are not in a position to use BIOS anymore and the kernel console driver has not been initialized yet. The decompressor contains a very primitive screen output routine, so it can show the "Uncompressing Linux" message.

2.3.2 Chain loading

Most boot loaders are designed to boot only one operating system. LILO knows only to load Linux kernels and the DOS boot sector can load only DOS. If you want to select between several different operating systems, it is not likely that you find a boot loader that can load all of them. But at least every operating system is supposed to be booted by a 512-byte boot sector that can be loaded by the BIOS and there lies the key. Any sufficiently advanced boot loader supports chain loading.

If a boot loader loads a boot sector of another operating system, it is called chain loading. This boot sector can directly be loaded from the disk or partition or it can be loaded from a file. For the other operating system it should not make a difference whether its boot sector was loaded by the BIOS or by another boot loader. In reality a boot sector of a partition is normally loaded by a master boot record program. Also in this case, it should make no difference if this is loaded by a more advanced boot loader (such as LILO) instead.

The following chain loading scenarios are possible.

- Linux boot loaders can chain load almost any other operating system.
- Linux boot loaders can be chain loaded by boot managers of other operating systems. It is possible to boot Linux from the Windows NT boot manager or the OS/2 boot manager.
- Linux boot loaders can chain load Linux boot loaders as well. This may make sense on computers with several independent Linux systems installed, where each Linux installation has its own local LILO and these can be selected by a central instance of LILO to chain load them. Instances of LILO can even exchange command lines between them.

2.4 Configuring the boot loader

Both LILO and GRUB have a configuration file that specifies several menu options, each representing either a Linux kernel or a different operating system to boot. For each Linux kernel a command line and an initial RAM disk can be specified. Apart from syntactic details the contents of these configuration files look remarkably similar. But there is an essential difference:

- LILO reads its configuration file at installation time. Every time the configuration file, the kernel or any initial RAM disk is changed, the LILO installer must be rerun. At boot time the configuration file is not read. The second stage boot program of LILO does not know how to find files in the

file system. It relies on a map file to find data blocks of the necessary files. This map file was created by the LILO installer.

- GRUB and also SYSLINUX read their configuration files at boot time. You can install the boot loader once and just change configuration files, kernels and RAM disk images without trouble. The second stage boot program knows how to find files in the file system.

3 Example Installations

3.1 Example root file system and kernel

In my article "Getting Linux into Small Machines"[7] I described how to build a kernel and a root file system to put on a Linux diskette.

The relevant files are:

- `zImage` the Linux kernel image.
- `root.img.gz` the compressed root file system image.

You can download a boot disk image `myboot.img.gz` from my home page and extract these files from it, so you can try them out with various other boot loaders.

```
gunzip myboot.img.gz
mount -o loop myboot.img /mnt
cd /home/lennartb/myboot
cp /mnt/boot/zImage .
cp /mnt/boot/root.img.gz .
umount /mnt
```

In the following sections I will show how to install this kernel and a RAM disk with this root file system on a diskette using each of the boot loaders discussed in this article. Whenever possible I will provide a boot time choice between the RAM disk and an alternative root file system that could be on a hard disk partition. i have tried each of the diskettes, just to make sure they really work.

3.2 Linux Boot Sector

The oldest way to boot Linux is the kernel's own boot sector. This method has several limitations:

[7] http://www.xs4all.nl/~lennartb/linux.html

- It can load a Linux kernel only from a diskette with no normal file system on it.
- It does not support a kernel command line. Instead, several parameters can be set in the boot sector using the `rdev` program. At boot time there is absolutely no way to select any options.
- It does not support initial RAM disks. It supports a different type of RAM disk instead.

It cannot boot from overformatted diskettes (1680kB on a 3.5”HD diskette), but it has zero file system overhead.

If the kernel and RAM disk together are 1200kB or less you can `dd` the *exact same disk image* to both a 5.25” and a 3.5” diskette and they will both work. There are few if any boot loaders that can do *that*!

First transfer the kernel and the compressed RAM disk image to the diskette. As the kernel is less than 450kB in size, we can specify an offset for the RAM disk of 450kB (`seek` option to `dd`). This way the kernel and the RAM disk do not overlap on the diskette.

```
dd if=zImage of=/dev/fd0
dd if=root.img.gz of=/dev/fd0 bs=1k seek=450
```

We need one more thing to get this working: we need to set the RAM disk offset in the kernel. First calculate 16384 plus the offset you used with `dd`, in our case it is 16384+450=16834. Use this number in the following `rdev` commands:

```
rdev /dev/fd0 /dev/fd0
rdev -r /dev/fd0 16834
```

It is important also to set the root device to `/dev/fd0`, otherwise it won’t work.

We could have put the RAM disk image on a separate diskette. In this case we should not specify an offset with `dd` and the `rdev` commands must be as follows:

```
rdev /dev/fd0 /dev/fd0
rdev -r /dev/fd0 49152
```

When this diskette is booted, the RAM disk will be loaded after the kernel initialization messages are shown, as opposed to the initial RAM disk that will be shown in other examples, which will be loaded by the boot loader before the kernel starts initializing. This type of RAM disk can also be used with other boot loaders.

Though the Linux boot sector cannot load a Linux kernel from an overformatted diskette there is no such restriction for the RAM disk. It is important that you `rdev` the kernel to the correct diskette type. For a RAM disk on a separate diskette, use the following commands:

```
rdev /dev/fd0 /dev/fd0u1722
rdev -r /dev/fd0 49152
```

For the RAM disk on the same diskette there are several solutions:

- Format the diskette with 21 sectors per track, but write the kernel with 18 sectors per track. The three extra sectors per track are effectively wasted, but it works. Of course the RAM disk is written with 21 sectors per track. The following commands have been tested:

  ```
  fdformat /dev/fd0u1722
  dd if=zImage of=/dev/fd0H1440
  dd if=root.img.gz of=/dev/fd0u1722 bs=1k seek=600
  rdev /dev/fd0H1440 /dev/fd0u1722
  rdev -r /dev/fd0H1440 16984
  ```

- Change the boot sector program of the Linux kernel so it expects 21 sectors instead of 18. This should be simple, but I have not tested it.

- Put a small minix file system at the start of the diskette and boot the kernel with LILO from that minix file system. The rest of the diskette holds the RAM disk and is not used by the minix file system.

These bootable disks are limited to 21 sectors per track. The 24 sectors per track formats use fewer, but bigger sectors, which the boot loader does not understand. The absolute maximum amount of Linux to put on a single 3.5” HD diskette can be achieved as follows:

- Format the diskette in two parts, one for the kernel with 21 sectors per track, one for the RAM disk with 24 sectors per track. `superformat` can format a range of tracks so you can make that disk by using it twice with different parameters.

- Put the kernel with the modified boot sector in the first area.

- Put the RAM disk in the second area,

- Calculate the RAM disk offset with respect to the 24 sectors per track disk. Give appropriate `rdev` commands.

Success with your copy-protected Linux boot diskette!

3.3 LILO

LILO is one of the first boot loaders for Linux that was capable of booting from the hard disk, dating back to 1992[8]. It's still the market leader and over the years it was updated with new features such as a nice boot menu and the use of the new BIOS functions that eliminate the 1024 cylinder limit. The basic working principle of LILO has stayed the same. This means that most work is done by the installer and not by the boot time code. LILO is the only boot loader that supports booting from a RAID system.

All Linux distributions already have LILO installed, but not necessarily the latest version. In this example we will download the latest version of LILO[9]. At the time of writing, the latest version of LILO was 22.3.1. Unpack the source tarball and make. This is easy as long as you have a sufficiently recent version of `bin86` installed. The file `README.common.problems` tells you where to get it. We won't install LILO for now, but run it from the source directory where it was built.

If you have read the "Getting Linux into Small Machines"[10] article, the following may be familiar to you, but we will do a few things differently:

- We will install on an overformatted diskette. This is not necessary for the current kernel and initial RAM disk, but for larger files, this may become necessary.
- We will use the minix file system instead of ext2. This has less overhead. The booted kernel need not support minix file systems, the host system must.
- We will use a boot menu.

First create the file `lilo-initrd.conf` with the following contents:

```
prompt
timeout=100
boot=/dev/fd0u1722
map=/mnt/boot/map
disk=/dev/fd0u1722
geometric
install=/mnt/boot/boot.b
image=/mnt/boot/zImage
  label=linux
```

[8] In fact `shoelace` is even older, but who has heard of that?

[9] http://brun.dyndns.org/pub/linux/lilo

[10] http://www.xs4all.nl/~lennartb/linux.html

```
  root=/dev/ram
  initrd=/mnt/boot/root.img.gz
image=/mnt/boot/zImage
  label=noram
  root=/dev/fd0u1722
```

Note the following:

- Specify `/dev/fd0u1722` as the diskette device.
- Using this version of LILO and an overformatted diskette, I was unable to use the `compact` option. This option works for normally formatted diskettes. The speed penalty for not using this option is not as great as with LILO 21.4.
- Using the `prompt` and `timeout` options we cause the menu to appear immediately and the default option to be booted after 10 seconds.

Finally type the following commands to format the diskette, make a minix file system on it, copy the files and run LILO:

```
fdformat /dev/fd0u1722
mkfs.minix /dev/fd0u1722
mount /dev/fd0u1722 /mnt
mkdir /mnt/boot
cp zImage /mnt/boot
cp root.img.gz /mnt/boot
cp lilo-22.3.1/boot-menu.b /mnt/boot/boot.b
lilo-22.3.1/lilo -C lilo-initrd.conf
umount /mnt
```

Instead of `boot-menu.b` there is still the old-style `boot-text.b`, giving exactly the same interface as old LILO versions and `boot-bmp.b` to show a bitmapped screen. You can try all three of course.

3.4 GNU GRUB

GNU GRUB[11] is the boot loader of the GNU project. It was the first boot loader to use the new BIOS features to boot beyond 1024 cylinders and it had a menu interface before LILO. Also it could detect more then 64MB of RAM when other systems could not yet do this.

[11] http://www.gnu.org/software/grub/

In an ideal world all operating system kernels would conform to a certain standard so that any kernel could be loaded by any boot loader. The *Multiboot* standard is such a proposed standard, but very few operating systems conform to it (primarily the GNU HURD operating system). GRUB is designed to conform to the Multiboot standard. Besides Multiboot, it can boot Linux and BSD kernels and chainload anything else. For BSD kernels the latest features of the latest version are not supported, but Linux support is up to date.

GRUB is huge compared to both LILO and SYSLINUX. This is mainly because it can read most file system types by itself. Besides it serves as its own installer and it has many useful and less useful commands. You can recompile it with some file systems and other features configured out to save space.

Just for fun and to show how flexible GNU GRUB really is, we will install the Multiboot compliant GRUB Invaders game[12] as well. Skip this if you don't like it. Unpack the tar archive and just use the `invaders` binary.

Download the latest version of GRUB (0.92), unpack it and type the following commands in the source directory:

```
./configure
make
```

Create the file `menu.lst` with the following contents.

```
root (fd0)
title Linux with RAM disk
   kernel /boot/zImage
   initrd /boot/root.img.gz
title Linux without RAM disk
   kernel /boot/zImage root=/dev/fd0
title GRUB Invaders
   kernel /boot/invaders
```

Create a diskette and copy the usual stuff to it. The fact that we create a DOS diskette is arbitrary. We could have performed the same procedure with almost any other file system that Linux supports. The files `stage1` and `stage2` are components of grub that are traditionally installed in the directory `/boot/grub` on the boot device. The configuration file is `menu.lst` is also copied to this directory. As opposed to LILO, GRUB reads the configuration file at boot time.

```
fdformat /dev/fd0H1440
mkfs.msdos  /dev/fd0
mount -t vfat /dev/fd0 /mnt
```

[12] http://www.erikyyy.de/invaders/

```
mkdir /mnt/boot
mkdir /mnt/boot/grub
cp zImage /mnt/boot
cp root.img.gz /mnt/boot
cp invaders/invaders /mnt/boot
cp grub-0.92/stage1/stage1 /mnt/boot/grub
cp grub-0.92/stage2/stage2 /mnt/boot/grub
cp menu.lst /mnt/boot/grub
umount /dev/fd0
```

Now we only need to make this diskette bootable. There are three ways to do it:

- Boot from an existing GRUB diskette or hard disk and install GRUB onto the diskette using the GRUB command line.
- Use the GRUB shell (this is what we will do). This is a program that runs under Linux and emulates the GRUB command line. You can do everything but boot a kernel.
- Use grub-install, a non-interactive tool to perform the task from shell scripts.

Run the GRUB shell:

```
grub-0.92/grub/grub
```

In the GRUB shell type the following commands:

```
root (fd0)
setup (fd0)
quit
```

The `setup` command installs a boot sector program on the given device. Note that GRUB can install its own boot sector, but it cannot write, and hence cannot copy, the required files `stage1` and `stage2` to the target device. This must have been done before by an operating system.

After you have installed GRUB onto the diskette, you can copy other kernels to it and edit the `menu.lst` file as much as you like, without the need to reinstall. Only the file `stage2` may not be moved.

When you boot the GRUB diskette, you get a menu with three entries. You can do any of the following:

- Just select a menu entry and press ENTER.

- Press the `e` key over a menu entry and edit that entry. This is useful for the second entry (Linux without RAM disk) to edit the `root` device for that command.
- Press the `c` key to go to the command line. The command line is very powerful.
  - The `cat` command shows the contents of any text file. Example, the following command shows the `/etc/passwd` file on `/dev/hda1`:

    ```
    cat (hd0,0)/etc/passwd
    ```

    While you type the command, pressing the TAB key shows you which files are available, just as in the `bash` shell.
  - The kernel command enables you to select any Linux kernel lying around on the hard disk and boot from it. Example:

    ```
    kernel (hd0,2)/boot/vmlinuz root=/dev/hda3 read-only
    boot
    ```

    This boots a kernel on `/dev/hda3`. This works, even if you hosed your LILO.

3.5 SYSLINUX

SYSLINUX[13] is a boot loader specially suited for diskettes. It boots only from DOS formatted disks (could be hard disk partitions as well). It supports custom help screens to introduce the user into using the diskette. DOS diskettes have the advantage that they are not considered defective by most PC users and therefore they less likely thrown away or reformatted. Like GRUB, SYSLINUX reads its configuration file at boot time.

In theory SYSLINUX can be installed in a FAT partition on the hard disk and it can boot Linux kernels from there and it can chain load MSDOS and similar operating systems. However, SYSLINUX is not the natural choice for booting Linux from the hard disk.

First obtain the latest version of SYSLINUX. (version 1.75 at the time of writing). Unpack and run `make`. Note that you need a recent version of `nasm` to rebuild the boot loader from source and that the binaries are already included, so you do not really need nasm just to install this boot loader.

Create a text file `syslinux.cfg` with the following contents:

[13] http://syslinux.zytor.com

```
prompt 1
timeout 100
say Available options: linux and noram
label linux
kernel zimage
append initrd=root.gz
label noram
kernel zimage
```

This file specifies the usual options for loading a kernel with an initial RAM disk and a kernel without an initial RAM disk (where a root device should be specified on the command line).

Next format a disk and copy all relevant files to it. An overformatted disk can be used and an MSDOS file system *must* be used. The `syslinux` command installs the boot sector program.

```
fdformat /dev/fd0u1722
mkfs.msdos /dev/fd0u1722
cd syslinux-1.75
syslinux /dev/fd0u1722
cd ..
mount -t msdos /dev/fd0u1722 /mnt
cp zImage /mnt/zimage
cp root.img.gz /mnt/root.gz
cp syslinux.cfg /mnt
umount /mnt
```

You can copy other kernels and initial RAM disk and edit syslinux.cfg as much as you like without any form of re-installation, just as you can do with GRUB. The GRUB binary on the example disk was 95k while the SYSLINUX binary is only 7k. That's the difference.

3.6 LOADLIN

LOADLIN is a program that can boot Linux from DOS. Since the advent of Windows ME and Windows XP, it is no longer natural that PC users have DOS installed on their PCs. Windows versions up to and including Windows 98 came with a DOS version that was usable for LOADLIN. In my opinion LOADLIN is primarily of historical interest for this reason. The primary reasons to use LOADLIN were the following:

- Getting devices working under Linux after loading their DOS device drivers. Some PC devices (mostly sound cards) were supposed to be fully compatible with a standard product (that Linux had a device driver for), but only after they were initialized by their DOS device drivers. So you needed to run a DOS device driver first before Linux would be able to access that device. Therefore it made sense to boot Linux from DOS.

- Booting from a Linux CD without the need to create boot floppies. Bootable CDs are a comparatively recent invention and most BIOS-es did not support it before 1998. Without the ability to boot directly from the CD you either needed boot floppies or you could start Linux on the CD from DOS using LOADLIN.

- Installing Linux without the need to tamper with the hard disk. With LOADLIN it is not necessary to install anything into the MBR in order to get Linux booted from the hard disk. A loadlin entry could easily be added to the DOS boot menu in CONFIG.SYS.

  Linux could even be installed without repartitioning the disk. For this very purpose the UMSDOS file system was added to the Linux kernel. It still exists, though it sees very little usage today. The Linux file systems lived in the LINUX directory on the DOS partition and it contained special files to translate long file names into DOS 8+3 file names and to specify attributes such as users, groups, permissions and device files. Linux could be installed just by unpacking a bunch of ZIP files, just as one would install a large DOS application. LOADLIN made booting Linux just as easy as running a DOS application, hence it fit nicely into the DOS mindset.

Loadlin can be downloaded from Hans Lermen's home page[14]. Download the file `lodln16.tgz` and unpack it. We will use only the file `loadlin.exe`.

In order to run LOADLIN, we need DOS. Create a bootable DOS diskette with just the bare system and `COMMAND.COM` on it. For those of you who have either Windows NT/ME/2000/XP or no Microsoft OS at all, you can use Freedos[15]. Download the file `FDB8_144.DSK` (the rawrite image) and transfer it to a diskette

```
dd if=FDB8_144.DSK of=/dev/fd0
```

Next remove everything but `KERNEL.SYS` and `COMMAND.COM` from this diskette.

```
mount /dev/fd0 /mnt
cd /mnt
```

[14] http://elserv.ffm.fgan.de/ lermen

[15] http://www.freedos.org

```
rm -rf config.sys *.bat kernel32.sys kernel16.sys readme\
    docs fdos games install util
cd
umount /mnt
```

Now that we have a minimal DOS diskette, copy `LOADLIN.EXE`, the linux kernel and the initial RAM disk to it:

```
mount /dev/fd0 /mnt
cd /home/lennartb/myboot
cp loadlin.exe /mnt
cp zImage /mnt/zimage
cp root.img.gz /mnt/root.gz
umount /mnt
```

Next try to boot from the diskette. You should boot into DOS. As there is no `AUTOEXEC.BAT`, you will be prompted for date and time. Press enter twice. Now you should be at the `A>` prompt. Type the following line to boot Linux:

```
loadlin zimage initrd=root.gz
```

Of course you can put this line in a batch file or even in `AUTOEXEC.BAT`.

3.7 Where Can Boot Loaders Live

Table 1 summarizes where boot loaders can be installed.

- The Linux kernel boot sector can only be installed on a raw diskette with no file system, it cannot be installed on a hard disk and not on a diskette with a file system.
- GRUB is the only boot loader that can live both on a DOS file system and on a different type of file system (ext2, minix). LILO cannot live on a DOS file system, SYSLINUX lives on a DOS file system only.
- The Linux kernel boot sector and GRUB do not work on diskettes with a nonstandard number of sectors per track. LILO and SYSLINUX do boot from diskettes with 21 sectors per track. The 24-sector formats work with no boot loader.
- The overhead is estimated for a boot diskette. It consists of the file system overhead (indicated by FS) and the second stage boot program. For LILO we have to add the map file, for GRUB and SYSLINUX we have to add the configuration file. For LOADLIN we have to add DOS as well.

Boot Loader	DOS FS	Other FS	Overformatted FD	Hard disk	Overhead
Kernel boot sector	No	No	No	No	0
LILO	No	Yes	Yes	Yes	FS + 15k
GRUB	Yes	Yes	No	Yes	FS + 96k
SYSLINUX	Yes	No	Yes	Yes	FS + 8k
LOADLIN	Yes	No	Yes	Yes	FS + DOS + 32k

Table 1: Where boot loaders can be installed

4 RAM Disks

The Linux kernel needs to mount a root file system before it can execute any processes. For rescue diskettes and CD's it is often quite natural to load the root file system into a RAM disk at boot time and mount the root file system on the RAM disk. This has the following advantages:

- The RAM disk can be stored compressed on the disk, so more programs can be stored on a diskette.
- The diskette or CD can be removed from the drive after the RAM disk is loaded, freeing the drive for other purposes, such as restoring backups or loading other utilities.
- Programs load faster from a RAM disk than from a diskette.

4.1 Living without a RAM disk

For a Linux installation on a hard disk it is quite natural not to use a RAM disk, but for a Linux installation that is booted from a diskette (without depending on a root file system elsewhere) there is only one valid reason not to use a RAM disk: you don't have enough RAM for the RAM disk. As a rule of thumb, if you have less than 6–8MB of RAM. you will probably need to boot without a RAM disk, though it can be achieved with 4MB if you really strip things down.

If you boot from a diskette without a RAM disk, with the root file system on a diskette, the kernel will always prompt for an extra diskette, so the kernel and the root file systems can live on separate diskettes. The root file system cannot be compressed. It is of course possible to put both on a single diskette. In that case the boot diskette contains the root file system and in that root file system the kernel lives, as would be the case with most hard disk installations. On the boot diskette you need to install a boot loader that can live inside that file system, which rules out SYSLINUX, but both GRUB and LILO should do.

Installing Linux on a machine from diskette without a RAM disk is a tricky process, which involves the following steps:

- Boot from a diskette without a RAM disk. The diskette may not be removed from the drive.
- Partition the hard disk, initialize the swap partition with `mkswap` and create a file system on the hard disk.
- Copy the contents of the root file system from the diskette to the hard disk.
- Reboot and select the root device on the hard disk (that you had just copied). This time the diskette drive is free.
- Copy all your data to the hard disk partition.

4.2 RAM disk devices

A RAM disk is just another block device, except that the buffers are never written to a real device, so the data exists only in RAM. The standard kernel comes with 16 RAM disk devices. Initially no memory is allocated to a RAM disk, but buffers will be added dynamically as data is written to it. The maximum size of a RAM disk is 4MB by default and can be modified at boot time with the `ramdisk_size` command line option. In LILO you could add the following command to the configuration file to double the size:

```
append ramdisk_size=8192
```

Note that this specifies a maximum size, smaller RAM disks take less space.

See the kernel source file `drivers/block/rd.c` how the RAM disk is initialized. There are three possibilities:

- A RAM disk is not initialized at boot time by default. On a system that was booted without a RAM disk, you can put data into the RAM disk block device and mount it. The following commands create an ext2 file system and mount that:

  ```
  mke2fs /dev/ram2 2880
  mount /dev/ram2 /mnt
  ```

- A RAM disk can be loaded at boot time from a device. In this case the kernel checks if the data is a valid `gzip` data stream or a valid file system (only a few types allowed). Compressed RAM disk images are decompressed by the kernel.

- A RAM disk can be loaded at boot time from the memory area where the boot loader put the initial RAM disk image. It is loaded in almost the same way (the read routines pull the data from memory instead of a device), but the main initialization code will treat the initial RAM disk differently with regard to mounting.

4.3 Loading a RAM disk at boot time

The feature to load a RAM disk from a diskette at boot time has existed almost from the beginning (it existed in 1992). Since then, some features have been added or modified. It is possible to load a RAM disk image from the boot diskette at a specific offset and to load it from a different diskette.

There are two ways to specify the RAM disk parameters.

- Use the `rdev` command to set the boot parameters. See section 3.2 for details.
- Set the `ramdisk_start`, `load_ramdisk` and `prompt_ramdisk` parameters from the kernel command line. This is how to do it in LILO:

```
root=/dev/fd0
append ramdisk_start=0 load_ramdisk=1 prompt_ramdisk=1
```

It is possible to put a file system on the first part of the diskette (and install LILO there) and to use the rest of the diskette for the compressed RAM disk image. Use a size argument to the mkfs command to specify the number of blocks for the file system.

It is also possible to put a (non-compressed) file system on the boot diskette and load the entire boot diskette as a RAM disk at boot time. Even the kernel, the LILO second stage boot loader and the map file would be loaded into the RAM disk. Advantage is that the same diskette can be used with or without a RAM disk.

4.4 The initial RAM disk

The initial RAM disk is loaded by the boot loader. There are two uses for the initial RAM disk:

- The primary purpose of the initial RAM disk is mostly of interest to maintainers of Linux distributions. The initial RAM disk makes it possible to install a minimal kernel image on the hard disk without the file system or hard disk device drivers compiled in. At boot time these drivers are loaded as modules, not from the hard disk, but from an initial RAM disk. After

doing that, the kernel would switch to the real root device (the hard disk). Without an initial RAM disk you would need the hard disk drivers (there are many possible SCSI host adapters) and the file system drivers compiled into the kernel, so you either have a huge kernel or many different kernels to choose from.

Such an initial RAM disk needs a file system with the following:

- A `/dev` directory with some essential devices.
- Some directories for mount points.
- A shell with some binaries. The `busybox`[16] program would be all that is needed.
- The required kernel modules.
- A startup script that loads the required kernel modules.

- The initial RAM disk can also be used for rescue, install or special-purpose diskettes, much as you would use the other type of RAM disk. You simply do not use the feature that you can switch back to the real root device. This is the type of diskette described in my "Getting Linux into Small Machines" article.

There is no reason why you should not be able to install an initial RAM disk from a second diskette, but most boot loaders do not support this. This is a limitation of the boot loader, not of Linux. Using the `pause` command, GRUB can do this, as the following part of a configuration file shows:

```
title Linux with RAM disk on separate diskette
kernel (fd0)/boot/vmlinuz
pause Insert the second diskette.
initrd (fd0)/boot/initrd.gz
```

5 Making Diskette Images without Diskettes

So far we have been preparing boot diskettes by writing to real diskettes. This sounds like the most logical way to do it, but there can be reasons why we want to prepare an *image file* of a diskette without using real diskettes. Several reasons could be:

- Create a diskette image for a diskette drive you do not have.

[16] http://www.busybox.net

- Automate boot image creation (several diskette images for a distribution).
- Create a diskette image for bootable CD-ROM
- Create a diskette image for a PC emulator

Basically we could create a diskette image as follows:

- Create an all zero image file using `dd`.
- Create a file system onto the image file.
- Mount the image file using the loop option and copy all files to it.
- Install the boot loader onto the image file.

The last part is the trickiest especially for LILO. It's fairly trivial for SYSLINUX and using the `device` command it can be done with GRUB. There is also another trick for SYSLINUX and GRUB (it does not work with LILO):

- Start with an image file with just the boot loader installed and an empty file system and copy that image file each time you create another image. This image file may be extracted from a real diskette just once.
- Mount the image file using the loop option and copy all files to it.

On Timo's Rescue CD Page[17] there is a good explanation of how to create 2.88MB diskette images for a bootable CD-ROM, using all boot loaders. I could not explain it better. Of course these recipes apply also to other types of disk images.

6 Hard Disk Installation

For a booting Linux on the hard disk there are basically the following choices:

- Use an OS neutral MBR boot sector program (such as the one from DOS) and let it boot a Linux boot loader in a primary partition. Some such boot sector programs let the user select one of several operating systems at boot time.
- Boot a Linux boot loader (LILO or GRUB) directly from the MBR.

[17] http://rescuecd.sourceforge.net/

- Chainload a Linux boot loader through the boot loader of another OS (the NT Boot Manager). One could even chainload Linux from another instance of a Linux boot loader. There could be an instance of GRUB that is booted from the MBR and GRUB could chainload several instances of LILO, each on its own partition.

- Boot Linux from a diskette. The diskette may contain just the boot sector (it loads the second stage of the boot loader from the hard disk), the entire boot loader or even the boot loader plus the kernel.

- Boot Linux through DOS using LOADLIN.

Which option you use, depends on several factors:

- What other operating systems run on the same machine? Chainloading Linux from the Windows NT boot loader would make no sense on a machine without NT.

- What partitions are used on the disk? Is a RAID used? As far as I know, LILO is the only Linux boot loader that boots from a RAID.

- What is the relative importance of each operating system?

- Perceived reliability of each boot loader.

- Personal preferences.

One factor that may complicate boot loader installation is that each operating system has its own naming and numbering scheme for hard disks and partitions and so does GRUB. The BIOS assigns a number to each disk. The first disk is 0x80, the second disk is 0x81, etc. When you install LILO, things go without problems most of the time, but there are situations in which it may be necessary to specify the BIOS code of a drive.

The naming convention of hard disks in GRUB can be confusing:

- The whole hard disk (if you want to install GRUB in the MBR) is called `(hd0)`, `(hd1)` etc.

- Partitions are counted from zero, so partition 1 on the first disk is called `(hd0,0)`. Extended partitions are counted from 4, like they are counted from 5 in Linux.

- Disks are also counted from zero in the order that BIOS assigns them. On most systems `/dev/hda` corresponds to `(hd0)` (except on systems with only SCSI disks), but what Linux disk will correspond to `(hd1)`, depends

Disk	Linux	BIOS code (LILO)	GRUB disk name	DOS drive
IDE Primary Master	/dev/hda	0x80	(hd0)	-
Primary DOS	/dev/hda1	-	(hd0,0)	C:
Primary Linux	/dev/hda2	-	(hd0,1)	-
Extended	/dev/hda3	-	(hd0,2)	-
Logical DOS	/dev/hda5	-	(hd0,4)	E:
Linux swap	/dev/hda6	-	(hd0,5)	-
IDE Primary Slave	/dev/hdb	0x81	(hd1)	-
Primary DOS	/dev/hdb1	-	(hd1,0)	D:

Table 2: Disk and partition names (first example)

Disk	Linux	BIOS code (LILO)	GRUB disk name	DOS drive
IDE Primary Master	/dev/hda	0x80	(hd0)	-
Primary DOS	/dev/hda1	-	(hd0,0)	C:
Primary Linux	/dev/hda2	-	(hd0,1)	-
IDE Secondary Master	/dev/hdc	0x81	(hd1)	-
Primary DOS	/dev/hdc1	-	(hd1,0)	D:
First SCSI	/dev/sda	0x82	(hd2)	-
Primary DOS	/dev/sda1	-	(hd2,0)	E:

Table 3: Disk and partition names (second example)

on what you connect to the EIDE ports. If the primary slave is a hard disk, this disk will be `/dev/hdb` in Linux and `(hd1)` in GRUB. If the primary slave is a CD-ROM and the secondary master is a hard disk, the hard disk will be `/dev/hdc` in Linux and `(hd1)` in GRUB.

Table 2 shows the disk and partition numbers on a system with two hard disks, connected as master and slave to the primary IDE controller. The first disk contains two primary partitions (DOS and Linux) and two logical partitions (DOS and Linux swap). The second disk contains one primary DOS partition.

Table 3 shows the disk and partition numbers on a system with three hard disks, one on the primary IDE controller and one on the secondary IDE controller and one SCSI.

If you use FreeBSD as well, things get even more complicated. BSD divides a normal disk partition into several sub-partitions. Linux can only access them if support for BSD disk labels is compiled into the kernel. Needless to say that Linux, BSD and GRUB all have different names for these sub-partitions.

In GRUB you have to use GRUB partition names for files accessed by GRUB, but Linux partition names for arguments on the kernel command line. The following is a very typical `kernel` command line in GRUB:

```
kernel (hd0,1)/vmlinuz root=/dev/hda2
```

Finally on some (mostly older) systems, disk geometry may cause problems. As long as everything uses LBA, no problems are expected. If your BIOS does not support LBA, boot loaders using the BIOS have no way to reach beyond 1024 cylinders. `nuni` *may* be a solution in such a case, but most of the time you will need to create a small partition near the start of the disk and boot the kernel from there.

7 CD-ROM Installation

Most Linux distributions come on a CD-ROM and there are various rescue disk images available that you can burn to a CD-ROM. In the near future, most PC's won't even have a floppy drive, so the CD-ROM is the only choice.

Most modern BIOS-es support the El Torito standard to boot from a CD-ROM. There are two different ways to boot Linux from a CD-ROM:

- Emulated diskette images are used in most cases. One file on the CD-ROM is a diskette image file. Several diskette sizes are supported, but 1.44MB is the most common. The BIOS loads the first 512-bit block into memory as if it were the boot sector of a diskette. Next it redirects all INT 0x13 calls for the first floppy drive to that file on the CD-ROM. Generally a program that boots from a diskette and that only uses BIOS calls to access the diskette, will work without change when booted from an emulated diskette image. The boot loaders LILO, GRUB and SYSLINUX should just work when booted this way.

  As an initial RAM disk on a diskette is loaded by the boot loader and the boot loader uses BIOS, this will also work from an emulated diskette image on a CD-ROM. This does not apply to a RAM disk that is loaded by the kernel (the non-initrd type), because the kernel does not use BIOS.

  The LILO, GRUB and SYSLINUX boot diskettes described in this document should all work from CD-ROM, provided you create them on normal 1.44MB diskettes and not on overformatted 1.72MB diskettes.

  It is not possible to use overformatted diskettes on a bootable CD-ROM (21 sectors per track 82 tracks), but it is possible to use 2.88MB diskette images. Real 2.88MB diskettes and their drives are very rare, but as diskette images on bootable CD-ROM's this format has gained popularity.

- It is possible to boot a program without diskette emulation. ISOLINUX is a program that can be loaded in such a way. It comes with the SYSLINUX package and can find files everywhere on an ISO9660 format CD-ROM. It has a configuration file that is almost identical to that of SYSLINUX. Using

this method there is no practical limit to the number of different kernels or to the size of an initial RAM disk. Note: if you use an initial RAM disk larger than 4 megabytes (uncompressed), you must specify an appropriate `ramdisk_size` parameter on the append line in the configuration file.

As soon as the Linux kernel is booted, it runs with the RAM disk as the root file system. From there it is easy to mount the CD-ROM. The CD-ROM can be filled with utilities, packages to install or whatever. In theory it is even possible to mount the CD-ROM as the root file system, but then it becomes impossible to swap disks while the system is running.

Both types of bootable CD-ROM can be created with `mkisofs`. The resulting ISO image can then be burnt to a CD-ROM using `cdrecord` or another CD writing program, even under Windows. I've tried both types using CD-RW disks.

Let's start with an emulated diskette. In the example I used the GRUB diskette, as it was already formatted at 1.44M (the image from my home page uses LILO and it should work fine too). First create a directory tree for the ISO image.

```
mkdir iso
mkdir iso/boot
mkdir iso/data
```

Next copy some files to the `data` subdirectory and copy the diskette to the `boot` directory.

```
cp somefile iso/boot/data
dd if=/dev/fd0 of=iso/boot/boot.img
```

Now create the ISO image.

```
mkisofs -o emuboot.iso -b boot/boot.img \
        -c boot/boot.catalog -r iso
```

Finally burn it to a CD-ROM. Use the appropriate device ID.

```
cdrecord dev=0,1,0 -eject -pad -data emuboot.iso
```

This CD is bootable and from the booted Linux you should be able to mount the CD-ROM to access the data files. As my CD-ROM is on `/dev/hdd` and the RAM disk image does not include this device, I had to `mknod` it first:

```
mknod /dev/hdd b 22 64
mount -r -t iso9660 /dev/hdd /mnt
```

Next create a bootable Linux CD-ROM with ISOLINUX. First create the file `isolinux.cfg` with the following contents:

```
prompt 1
timeout 100
say Available options: linux and noram
label linux
kernel zimage
append initrd=root.gz
label noram
kernel zimage
```

In fact this file is identical to `syslinux.cfg` on the SYSLINUX diskette.

Remove the old boot directory from the `iso` directory tree and replace it with the `isolinux` directory containing the kernel, the RAM disk image and the boot files:

```
rm -rf iso/boot
mkdir iso/isolinux
cp zImage iso/isolinux/zimage
cp root.img.gz iso/isolinux/root.gz
cp syslinux-1.75/isolinux.bin iso/isolinux
cp isolinux.cfg iso/isolinux
```

Next create the ISO image and burn it to CD as usual:

```
mkisofs -o isoboot.iso -b isolinux/isolinux.bin \
        -c isolinux/boot.catalog \
        -no-emul-boot -boot-load-size 4 \
        -boot-info-table -r iso
```

Older BIOS-es cannot boot from CD-ROM. There are two solutions:

- Any 1.44MB El Torito boot image can do double duty as a diskette image. The DOS program `RAWRITE.EXE` or Linux `dd` can transfer such a file to a diskette and we can boot from the diskette with no problems.
- If the machine runs DOS and DOS can read the CD-ROM, LOADLIN should be able to boot Linux from it.

Most Linux distributions provide both RAWRITE and LOADLIN on the CD-ROM and some even provide a real boot diskette.

8 Conclusions

There exist many boot loaders for Linux and no single boot loader is the best in all circumstances.

I would recommend to use the following boot loaders:

- For general purpose hard disk installations, use either LILO or GRUB.
  - GRUB has more features for recovery, but these can be dangerous as well. Of course you can protect the command line with a password.
  - LILO comes standard with all distributions. If you just want to install Linux (maybe together with DOS/Windows), use LILO. On the other hand, if you install different Linux distributions, different operating systems and do frequent kernel compilation, GRUB is absolutely for you.
  - LILO can boot from RAID systems while GRUB can't.
- SYSLINUX is ideal for diskette installation. It is compact, once it is installed, you can just copy files to DOS disks and it supports overformatted diskettes.
- For CD-ROM installations we could use diskette images with SYSLINUX, but ISOLINUX may be worth a try.
- A diskette with just GRUB installed and nothing else, can be a great rescue tool for machines that went unbootable. Using the GRUB diskette, you can mix and match kernels and initial RAM disk images from other diskettes and you can boot any kernel on any hard disk partition.