

Using LILO - The Linux Loader

Aeleen Frisch

In general, the boot process on a microcomputer has three stages: the system's master boot record (MBR) contains the primary boot program which starts the boot process and loads a secondary boot program from the boot blocks of the active partition; this second boot program is what loads the actual kernel.

Linux provides LILO, the Linux Loader, which can function as either a master boot program or a secondary boot program. **lilo** is installed with a command like this one:

```
# lilo -C /etc/lilo.conf
```

The `-C` option specifies the location of LILO's configuration file. (The location in the preceding command is, in fact, the default location, and so the `-C` clause is redundant.)

The `lilo.conf` file specifies LILO's behavior for certain aspects of the boot process and also defines the kernels and operating systems that it can boot. The following excerpt from a `lilo.conf` file lists the most important entries—and the ones that you are most likely to want or need to modify:

```
# Wait 10 seconds before autobooting 1st entry.timeout=100
# Allow user to enter a boot command.
prompt
# Where to install/configure LILO
# (no partition #=MBR) [see below]
boot=/dev/hda
...
# Text file displayed before boot prompt.
message = /boot/boot.message
#
# Default kernel is the first one listed.
image = /vmlinuz
# Boot prompt command to boot this kernel.
label = linux
# Fix for Sony CD-ROM (post 1.1.72)
# Specifies parameters to pass to kernel,
# changing CD-ROM's compiled-in I/O address.
append = "cdu31a=0x340,0,"
#
# An alternate Linux kernel
image = /safe
# Its boot command
label = safe
append = "cdu31a=0x340,0,"
...
# A different operating system (DOS)
other = /dev/hdb1
# Use the D: drive boot loader.
loader = /boot/any_d.b
# Use this partition table.
table = /dev/hdb
# This is the corresponding boot command.
label = ddog
```

[I tend to install LILO both in the MBR and the Linux partition for maximum flexibility, by running a second LILO command using its `-b` option (which replaces the boot entry in the configuration file):

Using LILO - The Linux Loader

Aeleen Frisch

```
# lilo -b /dev/hda1 -C /etc/lilo.conf
```

This way, if I decide to remove LILO from the MBR, I'll be all set to switch over to the Linux partition version.]

The final section ("stanza") illustrates the format for booting a DOS partition on the second hard disk; it uses an alternate loader, any_d.b, which tricks DOS into thinking it's on the C: drive. There are also loaders provided for OS/2 on the D: drive and DOS on the B: drive (os2_d.b and any_b.d, respectively; chain.b is the default loader for other operating systems).

If the label for a stanza is omitted, it defaults to the final component of the image or other entry (for example, vmlinuz or hda1).

The entry for a DOS partition on the C: drive is simpler, looking something like this:

```
other = /dev/hda1 label = dos
table = /dev/hda
```

This stanza is actually what you need for any foreign operating system on /dev/hda1. There is one additional trick needed if SCO Unix is the operating system you want to boot. In order for LILO to successfully boot a SCO Unix partition, that partition must be the only active partition on the C: drive.

This means you will have to turn off the active (boot) flag on the Linux partition and turn it on for the SCO Unix partition, using the Linux fdisk or cfdisk, before trying to boot SCO Unix (in addition to running LILO to install the new configuration). Note that LILO must be the master boot loader in this case.

Note: You will need to rerun the LILO command to reinstall it every time you rebuild the kernel or change any relevant aspect of the disk partitioning scheme. If you forget to do this, the system will not boot and you'll have to boot from a floppy. You will also need to rerun LILO if you change the text of the boot.message file.

More Complex Booting Scenarios

Booting a Linux partition on the second hard drive is also possible. For this to work, LILO must be installed in the MBR of the system's boot disk, as well as the secondary boot program in the Linux partition itself—this is usually taken care of when you install Linux on the hard disk and will be assumed in what follows. In this case, the best way to proceed is in two stages:

First, set up a lilo.conf like this one:

```
boot=/dev/hdaroot=current
image=/vmlinuz
label=linux
other=/dev/hda1
unsafe
other=/dev/hda2
unsafe
...
other=/dev/hdb4
unsafe
```

Using LILO - The Linux Loader

Aeleen Frisch

Define all of the partitions on both hard disks in the same way; the unsafe keyword tells LILO not to read the boot blocks or the disk's partition table for that entry—it basically says, "Trust me and do what I tell you." Install this LILO configuration, and make sure that Linux is bootable.

Then, modify the file, changing entries for any bootable partitions on /dev/hda to their correct form and removing ones you don't need, and rerun the LILO command.

It is also possible to boot a Linux partition on each of two disks. The procedure for doing so is the following:

Decide which one will be the usual Linux boot partition and set up LILO to boot it and any other non-Linux operating systems on both disks. Create an entry like the following for the second Linux partition:

```
other = /dev/hdb2 label=eviltwin
unsafe
```

Create a boot.message file which tells you which Linux will be booted when you select the default option. Install this configuration into the MBR on the C: drive.

Create (or retain) another LILO configuration for the second Linux partition, this time including an unsafe entry for the first Linux partition if you want to (this again assumes that LILO is installed in that partition, which usually happens at upon installation of the OS). Make sure that this partition's boot.message file also lets you know where you are. Install this configuration into the Linux partition only—make sure that the boot entry specifies the partition and not the disk as a whole.

The boot sequence will then go something like this:

```
Welcome to gallant.Boot choices: linux (default; on C:), dos,
  eviltwin (Linux on D:), sco
boot: eviltwin
Welcome to goofus.
Boot choices: test (default; on D:), goodtwin (Linux on C:)
boot: [Return]
Loading test...
```

Given these selections, Linux will boot from the D: drive. What happens is the LILO from the MBR on drive C: runs first, and it then starts the boot program on the Linux partition on the D: drive—which is again LILO. That (second) LILO then loads the kernel from the D: drive. (Note that if you wanted to, you could just keep popping back and forth between the LILO programs on C: and D: ad infinitum.)

If you think this is pretty silly, then omit the prompt keyword from the LILO configuration file for the D: drive (as well as its image section for the Linux partition on the C: drive), resulting in a simple lilo.conf file on the D: drive:

```
install=/boot/boot.bboot=/dev/hdb2
root=/dev/hdb2
map=/boot/map
image=/vmlinuz
  label=linux
```

Using LILO - The Linux Loader

Aeleen Frisch

Once this is installed, selecting eviltwin at the initial boot prompt will immediately boot the Linux partition on the second hard disk.

LILO's -r Option

Sometimes it is useful to be able to run LILO for a disk partition mounted somewhere other than /. For example, if you have another Linux root filesystem mounted at /mnt, you might want to run LILO to install the kernel (currently) at /mnt/vmlinuz using the configuration file /mnt/etc/lilo.conf. LILO's -r option is designed for such a purpose. It sets the root directory location for the LILO operation to the directory specified as its argument and looks for all files relative to that point. Thus, for the scenario we've been discussing, the correct command is:

```
# lilo -r /mnt
```

The Boot.Message File

The boot.message file is displayed before the boot prompt is issued. Here is an example boot.message file:

```
Welcome to JAG Property of the Linux Guerrilla Hackers
AssociationComputational science is not for the fainthearted!
Our current boot offerings include:
  * linux (smaller test kernel--1.3.10 currently)
  * safe (Yggdrasil distribution)
  * hacked (do you feel lucky?)
  * ddog - guess what ... (on D:)
```

An effective file will list all the defined labels (but it needn't be this eccentric).

Restoring the DOS Master Boot Program

Should you ever need to, here is the procedure for restoring the DOS master boot program:

- Boot from a bootable DOS floppy.
- Run the command fdisk /MBR

The Linux-FT Bootmanager

The Linux-FT distribution replaces the entire LILO apparatus with its Bootmanager facility, a graphical menu-based, user-configurable utility by which you can select which kernel to boot. Its display looks something like this:

```
          Bootmanager
Name      Location
MULTIUSER :3/vmlinuz ro root=/dev/sda3 2
SINGLE     :3/vmlinuz ro root=/dev/sda3 single
FLOPPY    A:
INSTALL   :3/vmlinuz root=/dev/sr0 ramdisk=300 5
DOS       C:1
LUCKY     :4/test_kern ro root=/dev/sda4
```

The Location field indicates the kernel to boot. For example, the MULTIUSER item will boot the file /vmlinuz on the third disk partition of the current disk, using the indicated device as the root filesystem, booting to run level 2. The DOS item will boot partition 1 on the first hard disk (C:). The final item is one added for this system.

Using LILO - The Linux Loader

Aeleen Frisch

Customizing the Bootmanager is easy. Press ESC to override the automatic boot timeout, and then use the cursor and function keys listed at the bottom of the screen to edit the relevant fields. Hardware options applicable to all boot sequences may be entered into the designated field below the boot choices menu.

Introducing Linux Loadable Modules

Very recently (since version 1.2), the Linux kernel has supported loadable modules. As of this writing, the Debian and Linux-FT distributions use this functionality by default.

In this scheme, you build a minimal kernel and dynamically load modules providing additional functionality as required. Such an approach has the advantage that many types of system changes will no longer require a kernel rebuild; it also has the potential to significantly decrease the size of the kernel executable.

The modules package provides utilities for building, installing and loading kernel modules (including ones needed to build a kernel with module support). Running `make modules` after building a kernel will create the loadable modules files, and `make modules_install` will install them into the `/lib/modules/1.2.6` directory tree (where the final component denotes the kernel release level). The configuration file `/etc/MODULES` (or `/etc/modules`) lists modules to be loaded automatically at boot time:

```
sysviBCS
PPP
```

This file says to load the modules supporting the System V filesystem type, the Intel Binary Compatability facility, and the PPP facility.

The following utilities are used to manipulate modules:

depmod: Determines dependencies among modules. For example, the `ppp` module I selected earlier requires the `slhc` module to function. `depmod` creates the file `modules.dep` in the relevant subdirectory of `/lib/modules`. This utility may be run automatically at boot time or you may need to execute it manually after building modules.

modprobe: Loads a module as well as all of those that it depends on (usually used to load modules on boots).

insmod: Loads a module interactively.

rmmod: Unloads a loaded module from the kernel (provided it is not in use).

lsmod: Lists currently-loaded modules:

```
Module:          # pages:  Used by:iBCS          19
PPP              5
slhc             2      [PPP]
sysv             7
```

At this point, the loadable modules facility is in its infancy, and only a few modules are available, but this will undoubtedly be the way Linux kernels generally operate in the not-too-distant future (and it is the way many other Unix versions already work).