10.6. Building a Custom Kernel

If you are running Linux on a system with hardware or wish to use features not supported in the stock kernels, or perhaps you wish to reduce the kernel memory footprint to make better use of your system memory, you may find it necessary to build your own custom kernel.

Upgrading the kernel involves configuring desired modules, compiling the kernel and modules, and finally installing the kernel image. This is followed by a system reboot (with fingers crossed!) to load the new kernel. All of this is documented in the ``README'' file which comes with each kernel package. Further information can be found in the ``Documentation/'' subdirectory. A particularly helpful file there is ``Configure.help'' which contains detailed information on the available kernel compile options and modules.

The following is a sample session demonstrating the build of a custom kernel, version 2.0.36 on the Intel platform. While building a custom kernel is usually just a matter of configuring, compiling & installing, sometimes (usually in the case of new hardware) it is necessary to download additional driver software should your hardware not yet be supported by the kernel version you are compiling.

The first step in building a custom kernel is to download and install the kernel sources from either RPM (preferred) or from tarball. See Section 10.4 for details on obtaining the appropriate files.

Next, use the ``rpm'' utility (or ``tar'', as appropriate) to install the kernel source tree and header files. For example, to install the 2.0.36-3 kernel RPM files:

```
rpm -Uvh kernel-source-2.0.36-3.i386.rpm kernel-headers-2.0.36-3.i386.rpm
rpm -Uvh kernel-ibcs-2.0.36-3.i386.rpm
```

(If you are running Linux on a notebook, you would also likely install the ``kernel-pcmcia-cs-2.0.36-3.i386.rpm'' file, which provides power management features.)

After installing the kernel files, you should be able to find the new source tree in the ``/usr/src/linux/'' directory.

The next step is to download any additional driver files (if applicable) and install them in the new kernel source tree. For example, to add support for the Mylex DAC960 hardware RAID controller, I would download the driver software from the http://www.dandelion.com/ web site. Unfortunately, such driver software are usually only offered as tarballs and need to be installed using the ``tar'' utility. For example:

```
cd /usr/src/
tar zxvpf DAC960-2.0.0-Beta4.tar.gz
```

You should read the documentation provided with your additional driver software, if applicable. For example, the DAC960 driver includes a ``README'' file which gives instructions on where the newly downloaded files should be located, and how to apply the kernel patch:

```
mv README.DAC960 DAC960.[ch] /usr/src/linux/drivers/block
patch -p0 < DAC960.patch
```

The next step is to ensure your system's symbolic file links are consistent with the new kernel tree. Actually, this step only needs to be done once, so the following needs to be done only if you haven't compiled a custom kernel before:

```
mail:/usr/src# cd /usr/include
mail:/usr/include# rm -rf asm linux scsi
mail:/usr/include# ln -s /usr/src/linux/include/asm-i386 asm
```

```
mail:/usr/include# ln -s /usr/src/linux/include/linux linux
mail:/usr/include# ln -s /usr/src/linux/include/scsi scsi
```

Note: Note: The above step is no longer necessary for 2.2.x or higher kernel versions.

The next step is to configure your kernel settings. This is the most important step in building the custom kernel. If you disable the wrong settings, you may leave out support for features or hardware you need. However, if you *enable* the wrong settings, you will be needlessly enlarging the kernel and wasting your valuable system memory (that being said, it is probably better to err on the side of the latter rather than the former).

The best way of ensuring you compile the kernel properly is to know what features you will need to use, and what hardware is in your system that you will require support for. After you have gained experience in customizing your kernel a few times, the process will become "old hat" and won't seem so intimidating!

Type the following to begin the configuration process:

```
mail:/usr/include# cd /usr/src/linux
mail:/usr/src/linux# make mrproper
mail:/usr/src/linux# make menuconfig
```

(You could type ``make xconfig'' instead of ``make menuconfig'' if you have the X Window System running; see Chapter 5 for details on how to get X working.)

To configure your kernel, go through the various settings and select (enable) whichever ones you require, and de-select (disable) the ones you do not require. You can choose between having such support built right into the kernel, or having it built as a module which is loaded and unloaded by the kernel as needed. (If you compile a feature that is actually needed to boot your system, such as a SCSI driver, as a module, you will need to create a RAMdisk image or your system will not boot. This is done with the ``mkinitrd'' command; this procedure is described a little further down.)

When going through the configuration settings, you can select <Help> for a description of what a given kernel option is for.

After you have configured your kernel settings, type the following commands to compile your kernel:

```
mail:/usr/src/linux# make dep ; make clean
mail:/usr/src/linux# make bzImage
mail:/usr/src/linux# make modules
```

If you are *recompiling* the same kernel as you have previously (2.0.36-3 in this example), you will likely want to move the existing modules to a backup directory as with the following command:

```
mail:/usr/src/linux# mv /lib/modules/2.0.36-3 /lib/modules/2.0.36-3-backup
```

Now, type the following command to actually install the new modules:

```
mail:/usr/src/linux# make modules_install
```

The next step is to copy the kernel into the ``/boot/'' directory and use LILO to update the boot record so that the new kernel is recognized. The following commands will make a backup copy of your existing kernel, copy the new kernel over, and then refresh the LILO boot record:

```
mail:/usr/src/linux# cd /boot
mail:/boot# cp vmlinuz vmlinuz.OLD
mail:/boot# cp /usr/src/linux/arch/i386/boot/bzImage vmlinuz-2.0.36
mail:/boot# /sbin/lilo
```

Finally, you will need to edit your ``/etc/lilo.conf'' file, and make sure the "image" reference is pointing to the new kernel. You should also add a section which points to your backup kernel, called, perhaps,

"OldLinux". Here is an example file:

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=50
image=/boot/vmlinuz
        label=Linux
        root=/dev/hdb1
        read-only
image=/boot/vmlinuz.OLD
        label=OldLinux
        read-only
```

By adding your backup kernel information in this way, should your new kernel fail to boot properly (perhaps a device is not recognized, or a daemon doesn't start as it should), you can simply type ``OldLinux'' to boot from the old kernel and investigate the problem.

> **Note:** Note: As mentioned previously, if you've compiled a feature required to boot your system as a module, you will need to create an initial RAMdisk image in order to boot your system. (Don't forget to compile your kernel with support for such an initial boot image.)

The procedure to create and use an initial RAMdisk image is as follows:

- Add an entry in your ``/etc/lilo.conf'' to boot off the initial RAMdisk image; this is shown as an addition to the example configuration file shown earlier:

  ```
  image=/boot/vmlinuz
          label=Linux
          root=/dev/hdb1
          initrd=/boot/initrd-2.2.4-4.img
          read-only
  ```

- The loopback device needs to be loaded before you are able to use the mkinitrd command. Make sure the loopback device module is loaded:

  ```
  /sbin/insmod loop
  ```

  (If you get an error message about not being able to load the loopback module, you may need to specify the full path to the module for the *current* kernel your system is still running on, for example ``/lib/modules/2.0.35/loop''.)

- Use the ``mkinitrd'' command to actually create the image:

  ```
  /sbin/mkinitrd /boot/initrd-2.0.36-3.img 2.0.36-3
  ```

- Run ``/sbin/lilo'' to update your boot loader.

Now, shut down your system and boot the new kernel!

```
mail:/boot# /sbin/shutdown -r now
```

If your kernel refuses to boot altogether, don't panic. Boot off the boot disk that was created during the installation of Linux . If you don't have copies of this disks, you should be able to create one from the Red Hat CD. Insert the boot diskette into the drive and reboot the computer. When you see the "boot:" prompt, type:

```
mount root=/dev/hda1
```

The above command assumes your "/" (root) partition is located on /dev/hda1.

Linux should then boot normally (although since you are using the kernel from the boot disk, not all services or devices may operate properly for this session), and then you can restore your old kernel and reinstall the LILO boot loader information (ie. ``mv /vmlinuz.old /vmlinuz ; /sbin/lilo'') and shutdown/restart. You can then try recompiling the kernel with different options and try again.

Prev Upgrading a Red Hat Stock Kernel | Home Up | Next Moving to the Linux 2.2.x Kernels