Installing the kernel 2.6 – a step-by-step guide

Chief Installer

Improved support for multimedia applications, enhanced system performance, better hardware support, and the new IPsec security module to improve system security. There are many good reasons to start using the latest kernel now. We show you how with a step by step guide.

BY EVA-KATHARINA KUNST AND JÜRGEN QUADE

Five million lines of code packed in a 32 MByte archive: the new kernel's figures are certainly impressive [1], although configuring and installing the kernel is certainly non-trivial. That said, if you have some experience of compiling and configuring the 2.4 kernel, you should have no difficulty finding your way around in 2.6. A word of warning to kernel newbies, however. A kernel update is a far-reaching system modification that will punish any unconsidered steps.

The following article points out a direct path to a functional kernel 2.6. Tuning the kernel to reflect your system environment requires a bit more patience. For additional information, refer to the Post-Halloween Document by Dave Jones at [2] (a useful resource) and also the kernel mailing list archive at [4]. In contrast to many projects on this scale, no special tools are required to build your own kernel. Useable versions of the required ingredients – a compiler, an assembler, and make – are available on nearly any Linux system. If in doubt,

THE AUTHOR

Eva-Katharina Kunst is a journalist, and Jürgen Quade a professor at the Hochschule Niederrhein. Both have been keen Open Source supports since the early days of Linux.

Kernel 2.6 under Suse

SuSE 9.0 is kernel 2.6 ready. A 2.6 test kernel is supplied with the distribution, and newer versions will be available at [7]. However, the */usr/sbin/hwbootscan* may cause some issues in combination with a 2.6 kernel. You should disable the program in this case. To do so, disable the *HWBOOTSCAN_BIN* variable in line 20 of the */etc/init.d/hwscan* script as follows.

```
# HWBOOTSCAN_BIN=➥
/usr/sbin/hwbootscan
```

The kernel 2.6 supplied with the distribution makes it easier for you to build your own kernel, as the existing configuration can be used as a template. Use *rpm* to install the test kernel (YaST cannot do this). Then copy the 2.6 kernel configuration file to the directory with the current Linux sources:

```
cp vmlinuz-2.6.0-test5-5-➥
default.config /usr/src➥
/linux-2.6.1-test11/
```

After copying the file, re-run *make menuconfig*. This simply adds any new options automatically to the kernel configuration. For this, and the following steps, ensure that the developer tools, GCC, make, ncurses and Qt3-devel for *make xconfig* are installed After configuring the kernel, follow steps 5 through 8 as defined in the text. You may need to edit */etc/modprobe.conf* after loading some modules.

Suse stores additional entries in the */etc/modprobe.conf.local* file. The syntax for this configuration file is described in the manpages (*man modprobe.conf*).

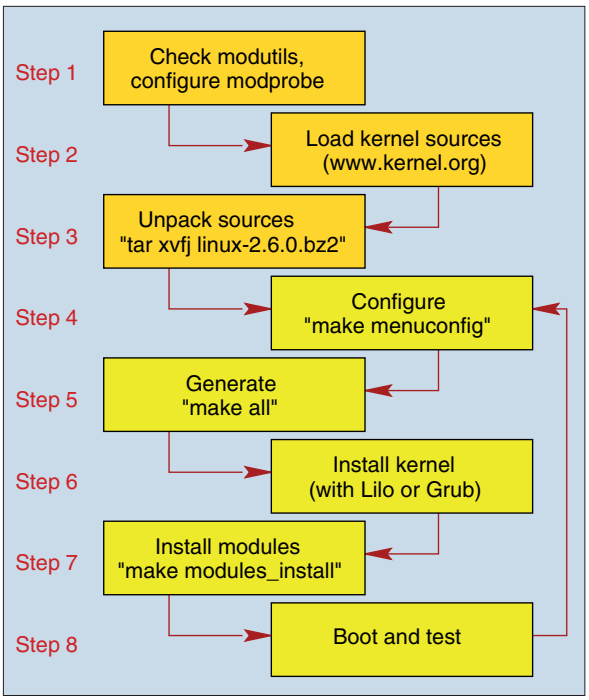

Figure 1: Eight steps to your own kernel. Kernel newbies may need to get used to performing steps 4 through 8 more than once.

you can install the developer packages. It does not really matter whether you use GCC version 2.95, 3.2, or 3.3, and make version 3.80 to compile and build the kernel. Kernel 2.6 is fairly insensitive to version numbers. When we compiled the kernel on our Debian system, we were equally successful with all three compiler versions.

In contrast, modutil dependencies need to be taken seriously (step 1 in Figure 1). The modutils are responsible for loading and unloading the kernel modules, but the Linux developers have completely re-designed this mechanism. Kernel 2.6 modules need the latest version of the *insmod* to load.

The new insmod will call the older variant if necessary. This explains why the older *insmod.old* version needs to be kept. You can easily find out whether a 2.6 aware version of this program is installed. If you find a program called *insmod.old* below */sbin/* or */usr/local/sbin*, *insmod* should be able to handle the new kernel. Fortunately, many current distributions take care of this (for example Suse 8.2 and 9.0). If this is not the case, Debian (or Knoppix) users can launch *apt-get* to install the new modutils.

```
apt-get install ➥
module-init-tools
```

Non Debian systems can download a current version of the modutils from [3], for example 3.0-pre5. Expanding the program package will create a *module-init-tools-3.0-pre5* directory that contains a *README* file with a slightly terse description of the most important details. For one thing, the file tells you that you will need to run the following commands:

```
./configure --➥
prefix=/
make moveold ➥
# Only for the first➥
round!
make
make install
```

The original program, */sbin/insmod*, is moved to */sbin/insmod.old*. The new */sbin/insmod* calls the previous version, if your system boots a kernel 2.4, but follows the new specification when loading kernel 2.6 modules. The same applies to the */sbin/rmmod* and */sbin/modprobe* programs.

New Configuration Files

The new modutils require a new configuration file: */etc/modprobe.conf*. If this file is not available on your machine, you should use *./generate-modprobe.conf /etc/modprobe.conf* to create it. If you are a fan of the device filesystem, you will also want to copy *modprobe.devfs* to the */etc/* directory.

After fulfilling these requirements, you can continue with step 2 (see Figure 1). This means retrieving the 2.6 kernel sources off the Web. The welcome page of the kernel archive [1] lists the current versions. To access the full sources rather than a patch, ensure that you click on the *F* (for Full), which is hiding on the right of *2.6.1-test11* (see Figure 2). Store the sources in the */usr/src* directory (Step 3):

```
cd /usr/src
tar xvfj /tmp/➥
linux-2.6.1-
test11.tar.bz2
```

The most difficult and time-consuming step (number 4) follows immediately after unpacking the archive: the kernel configuration. In our opinion, *make menuconfig* is best suited to this task (see Figure 3). The command creates and launches an ncurses-based configu-

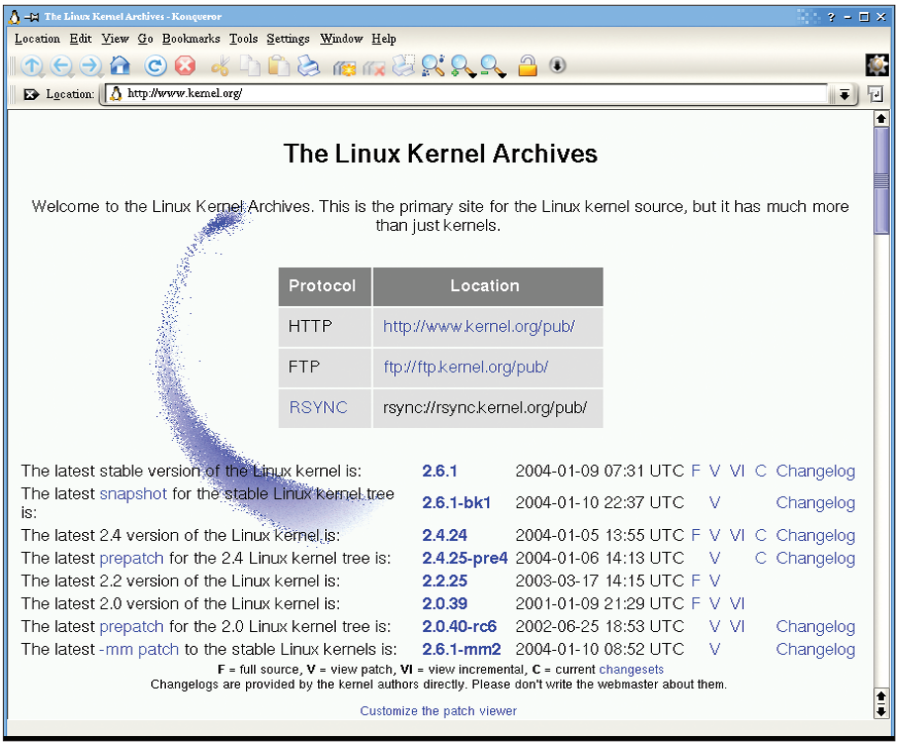

Figure 2: The kernel archive has a complete collection of kernel versions. The welcome page has a link to the current kernel. Previous versions are accessible via the links in the table at the top.

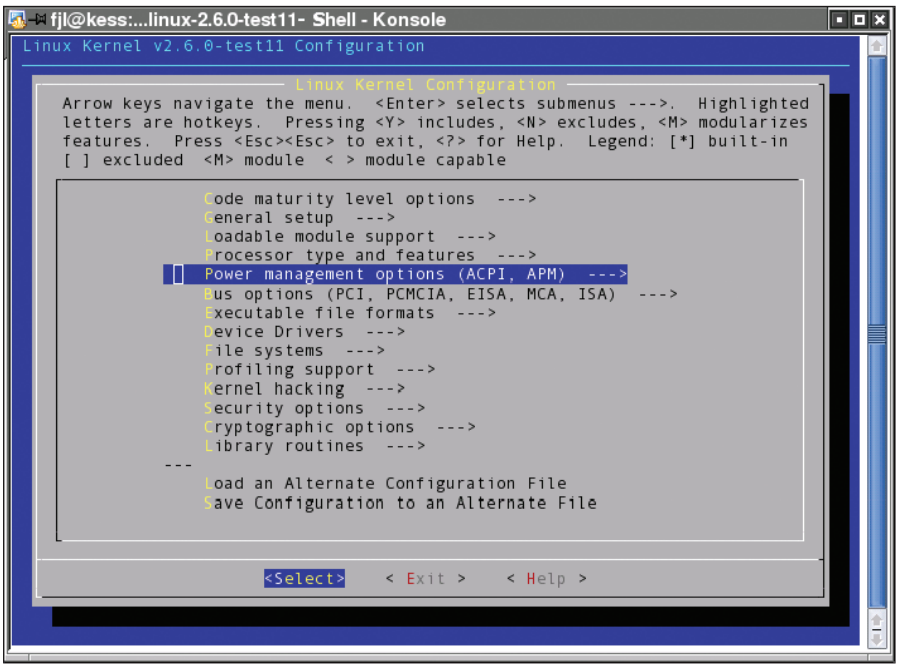

Figure 3: Configuring the kernel has become a lot more understandable despite the number of options available for 2.6. The picture shows the *menuconfig* configuration tool at work.

ration program that you can run in any normal console window. A graphical variant based on Qt is also available (see Figure 4): To use this version, enter *make xconfig*.

Patience

Despite a mass of new options, the configuration process has become more intuitive. Make sure that you change the default configuration for the processor type (*Processor type and features* menu) and the accompanying system architecture. Watch out for filesystems: If you are a Suse user, you will definitely need *ReiserFS support*. If you do not need specific components (such as the SCSI subsystem), you should disable them.

Help is available for almost every point. The first time you work your way through the menu, the process can take a good two or three hours. But that is nothing in comparison to the more than 5 million lines of code that this process configures.

If you have a previous working 2.6 kernel configuration (see the "Kernel 2.6 under Suse" box), you should base your new configuration on it. Otherwise, go for a minimalist approach: Select only those components that you require urgently. Once you have the kernel running, you can add new features and drivers at any time later.

Generation

After storing your settings in the configuration file, you can go on to generate the kernel. Debian or Knoppix users should refer to the "Steps 5 through 7 on Debian" box.

All other users should enter *make all* in the source directory (step 5). You can ignore any warnings displayed on screen. On a system with a lot of memory, and a 2.6GHz Intel processor, you can expect to see the prompt again in about 10 minutes.

If the kernel generation process quits unexpectedly, you will need to inspect the output to discover the cause and then re-configure the kernel (by calling *make menuconfig*). Ensure that you disable the troublesome component.

Steps 5 through 7 on Debian

The unstable version of Debian (Sarge) offers a number of kernel 2.6 downloads. But tuning and building a kernel that will provide optimal support for your system does require a few manual steps. After completing the configuration (step 4) call *make-kpkg* in the source code directory:

```
./make-kpkg kernel_image
```

This script generates the kernel and modules (step 5) and creates the Debian *kernel_image* package in a directory below the source directory (typically */usr/src/*). Before you install this (steps 6 and 7), you should add a new option to your boot loader. When you run *dpkg -i kernel-image-Kernelversion.deb* to install the image package, the Debian installer also launches *lilo*. This should introduce lilo to the new kernel, and allow the boot loader to display this boot option.

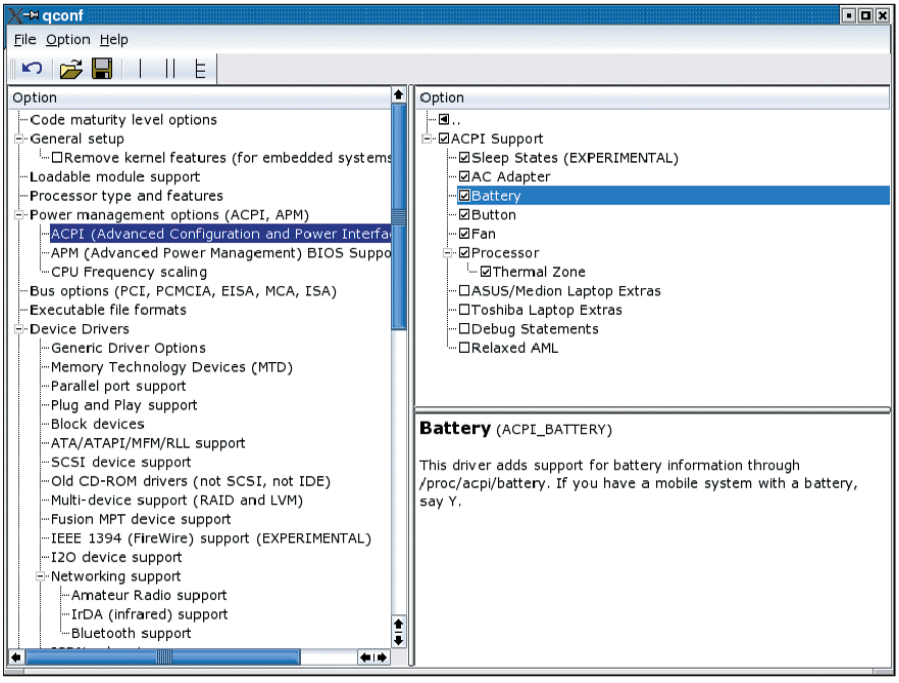

Figure 4: The Qt-based *xconfig* provides a useful overview. The configuration areas are shown on the left, the individual options at the top, and below them the help texts for the current parameter.

Installation

If everything worked out fine, the installation – steps 6 and 7 – are next. Installing the modules is easy. Just call *make modules_install*, and the modules will appear in a new directory called */lib/modules/2.6.1-test11/* a few minutes later. But you will need to copy the kernel manually to your */boot/* directory:

```
cp /usr/src/linux-2.6.1-test11
/arch/i386/boot/bzImage /boot
/vmlinuz-2.6.1
```

The boot loader should have an option for the new kernel when you fire up your machine. For lilo [5] this means editing */etc/lilo.conf*, and for grub [6] the */boot/grub/menu.lst* file. Add a new entry with the name of the kernel (for example */boot/vmlinuz-2.6.1*) and a symbolic name for the menu item. You only need to do this once.

If you later add a new driver to the kernel, it is typically unnecessary to edit *lilo.conf* and *menu.lst*. If you use the lilo boot loader, don't forget to run *lilo* when ever you install a new kernel.

Test Boot

So let's boot the machine. If the kernel fails to complete the boot process, you will have to evaluate the output on the console. These messages almost always provide the reason why the kernel has failed to boot. In many cases, a required driver will be missing, and that could prevent access to the hard disk, or the root filesystem, for example.

If the screen simply stays blank, troubleshooting may be tricky. This is typically an issue with the framebuffer device. There are two things you can do:

- Remove the *vga=XXX* option from */etc/lilo.conf*, or enable *vga=normal*.
- Check whether the *CONFIG_VT* option is set to *y* in the kernel configuration file, */usr/src/linux-2.6.1-test11/.config*.

If you succeed in booting the new kernel, you may need to create a few device drivers for any of your components that the default kernel does not support. nVidia graphics adapters are just one example. For more information see [8].

Check the boot messages next. Running *dmesg* allows you to review older messages. If some issues are found, go back to step 4. If not, you have a working kernel 2.6. Well done! ■

INFO

[1] Kernel sources: *http://www.kernel.org*

[2] Dave Jones, "The Post-Halloween Document": *http://www.linux.org.uk/~davej/docs/post-halloween-2.6.txt*

[3] New modutils: *http://www.kernel.org/pub/linux/kernel/people/rusty/modules/*

[4] Linux Kernel mailing list: *http://lkml.org*

[5] Lilo: *http://lilo.go.dyndns.org/*

[6] Grub: *http://www.gnu.org/software/grub/grub.html*

[7] Kernel 2.6 packages for Suse 9.0: *ftp://ftp.suse.com/pub/suse/i386/9.0/unsorted/kernel-2.6/*

[8] nVidia kernel modules: *http://www.minion.de/*