Rebuilding the kernel for non-experts

KERNEL BUSINESS

Worried about a recent security exploit? Want to take advantage of a new hardware feature? You don't need to be a Linux expert to patch and compile the Linux kernel. We'll show you how to get started.

BY PETER KREUSSEL

The kernel allocates memory to applications, controls access to the filesystem, and performs many other important tasks, but for most users, the kernel is always hiding behind a shell or a GUI. Ordinary users usually don't need to tinker with the kernel.

The Linux kernel, however, is actually quite visible – it resides in the */boot/vmlinuz-kernelversion* file. Depending on your distribution, the kernel can occupy 1 or 2MB on disk. But this single file is only part of the picture. The files below */lib/modules/[kernelversion]* are just as much a part of the kernel. This directory weighs in at no less than 70MB on Suse 10.0, but not everything in this directory requires memory space. In fact, the modules in this directory are just loaded on demand.

More or less every distribution comes with ready made binaries for the kernel and modules, however, it is possible to compile the kernel just like any normal program. To do so, you need the kernel sources and the GNU C Compiler.

Time for Something New

Distribution-specific kernels normally work without causing much trouble, but in some situations, you may find yourself needing to build a new kernel. For instance, if you have security-critical applications, you may want to rebuild the kernel if vulnerabilities are disclosed. And in some cases new kernel versions have improved support for new hardware.

You may also have occasion to work with kernel patches. Patches give you the ability to add new functions to the kernel. One example of this is Software-Suspend2 [1]: this patch, which is really useful for laptop users, gets Suspend working on systems where the plain vanilla kernel fails.

Table 1: Kernel Functions

Resource management (memory, CPU cycles etc.)
Filesystem access
Network access
Access to hardware components

Keeping track of kernel development is not a trivial task because the changes between versions are just too numerous. The changelog on Kernelnewbies.org [2] gives you an overview. If you are rebuilding the kernel to address a hardware problem, you are more likely to find notes about the problem in the context of the hardware device you are trying to fix. If a forum tells you that your DVB card will work with kernel 2.6.13, but you have 2.6.12 on your system, it might be time to fire up your compiler.

Building a new kernel is not a risky process. The bootloader will let you choose between different kernels. After installing a new kernel, there is nothing to stop you booting the older version. Figure 1 shows the steps for getting a do-it-yourself kernel build running on your machine.

Kernel Sources

Your distribution might have a package with updated kernel sources, but the latest available kernel will always be available at Kernel.org. As the site is often hit by capacity problems, make sure you use a mirror [3]. Download the kernel

Table 2: Kernel Configuration Main Menu

Menu entry	Function	Default setting
Code maturity level options	Allow selection of experimental drivers	Enabled
Processor Type and features	Processor architecture settings	Normally, only the *Processor family* will need modifying.
Power management options	Useful for laptop owners	Can be enabled without any danger
Networking	For experts only (e.g. Bluetooth and infrared support)	Keep the defaults
Device Drivers	Configure drivers for hardware devices	Modify as required
File systems	Support for filesystem types	May need modifications if you are doing without an initial ramdisk

with the highest version number, and unpack the archive in */usr/src*.

You should be aware that the "plain vanilla" kernel could differ considerably from your distribution kernel. Distributors typically add a large number of patches that add functionality, improve driver support, and resolve known bugs.

The question as to whether the plain vanilla kernel from Kernel.org or the distribution-specific version provides better underpinnings for a stable system is mainly a question of faith. Neither is likely to cause major issues. This said, if you want the most recent version of the kernel, you may have no alternative to the official version.

This article works with the official release by the kernel developer team. An alternative is the so-called *mm* tree, which contains contemporary enhancements from the kernel development process. These enhancements can be both bug fixes and extensions, which need to go through final testing before being adopted into the standard kernel. Andrew Morton releases the *mm* tree as a patch against the standard kernel. The "Patching the Kernel" box describes how to apply a Linux kernel patch.

Spoiled for Choice

The Linux kernel has a modular structure: various functions (for example support for a specific filesystem or a specific sound card) can be enabled or disabled when you build the kernel. The more functions you enable, the bigger the kernel will become – at least in theory. The compilation time will be longer if you use the kernel's on-demand load function.

Most kernel features can be added to the kernel as modules or built in. Only the built-in components will be resident in memory as Linux loads kernel modules on demand. This is why it does not make much difference in practical terms if you enable kernel functions you may not need (Figure 2).

Patching the Kernel

Copy the patch to */usr/src/* and unpack patches ending in *.gz* with *gunzip*. Then change to the kernel directory and use the same patch tool to apply the patch:

```
$ cd linux-2.6.[Version]
$ patch -p1 <../[Patchfile]
```

Preparation

As Linux distributions are designed to run on almost any kind of hardware, it is common to include almost everything the kernel has to offer in the form of modules, with the exception of some extremely rare or problematic components.

Most distributions store the standard kernel configuration in */boot/config-kernelversion*. Root can query the current kernel version by entering *uname -r*. Suse users will not find the kernel configuration in */boot*; instead the current kernel dynamically generates a *config.gz* file below */proc*. Copy this file to the kernel directory, run *gunzip config.gz* to unpack the file, and rename it *.config*.

Run all *make* commands as root and from within the */usr/src/linux-kernelversion* directory. Start by removing the traces of the development process from the kernel tree you just unpacked by entering *make mrproper*. Then copy the configuration file to the kernel tree by entering *cp /boot/config-kernelversion /usr/src/linux-kernelversion/.config*, where *kernelversion* is the version of the Linux kernel you are building.

The kernel includes a Qt-based, a Gtk-based, and a console-based configuration front-end. These tools help you select the functions your do-it-yourself kernel build will provide. The configuration menu, which has the same entries no matter which front-end you choose, contains several hundred settings – a daunting prospect at first glance.

You just copied the configuration files, and you will have a pre-configured kernel after running the configuration tool by entering *make menuconfig* (console version), *make gconfig* (Gtk version), or *make xconfig* (Qt version) (Figure 2). The console version requires *ncurses-devel* and is generally regarded as the most reliable tool.

If you copy a *.config* file from an older kernel version to your source tree, make sure you run Menuconfig or another

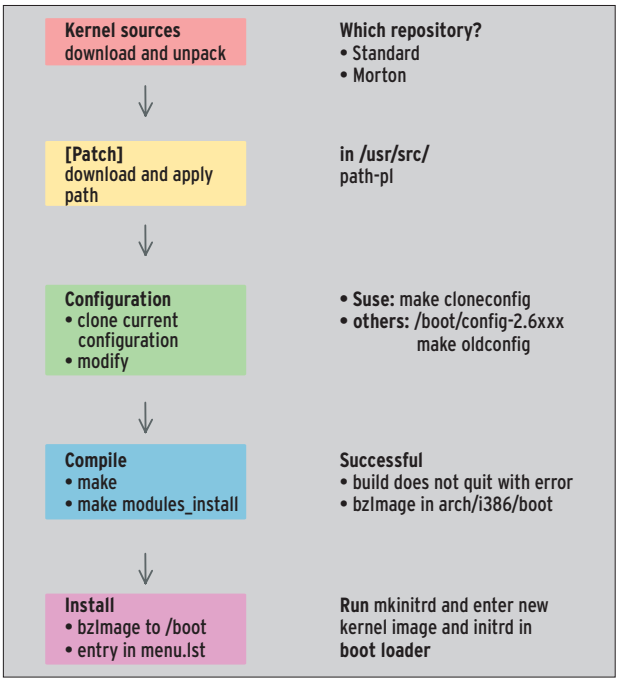

Figure 1: Step by step to the new kernel: you don't have to be an expert to build your own.

configuration tool at least once before you jump into the build. This will add configuration options new to the latest kernel version and automatically set the defaults.

Now quit the configuration without changing any settings. To quit the configuration, select *Exit* in Menuconfig or Xconfig until the tool asks you if you would like to save the configuration. Say yes when prompted. If you are working with Gconfig, you will have to save the configuration yourself; failure to do so will result in any changes being discarded.

After creating the new kernel and booting once from the it, launch the configuration tool again, and modify the kernel configuration once more if necessary.

Home Straight

Now the only thing separating you from a ready-to-run kernel is a *make* (and a wait of between 5 and 30 minutes). During the build, you will see many messages output on your screen. You can typically ignore warnings (Figure 3). There are a few exceptions for Debian and Ubuntu users (see box).

If the build completes without the dreaded word *error* appearing on your screen, you will find a new kernel in the kernel source code tree below */arch/processor-architecture/boot* in the form of the *bzImage* file. For a PC, this is */arch/i386/boot*; this also applies to *amd_64* kernels. Copy the *bzImage* file to the */boot* directory. Rename the file to *vmlinuz-kernelversion*. Also copy the *system.map* file to */boot/system.map-kernelversion*. Then call *make modules_install* to install those parts of the kernel you configured as modules.

All you have to do now is add an entry for the new kernel to the bootloader start menu. If your distribution uses the Grub bootloader, and most distributions do nowadays, you can edit the */boot/grub/menu.lst* file to do this. If you have Lilo, the entries are in */etc/lilo.conf*.

Building the Kernel on Debian and Ubuntu

Debian and Ubuntu users should not use the *make* command to compile the kernel. They should use *make-kpkg* instead. This command assumes you have the *kernel-package*. If you need an initial ramdisk, add the *--initrd* parameter.

Up and Running

A classic chicken and egg problem occurs at boot time: to read filesystems, the kernel needs modules that reside within a filesystem on disk. Most distributions opt for a so-called initial ramdisk to resolve this. The kernel first temporarily mounts the initial ramdisk as the root filesystem, and the bootloader places the ramdisk's content at the kernel's disposal. The kernel finds the required

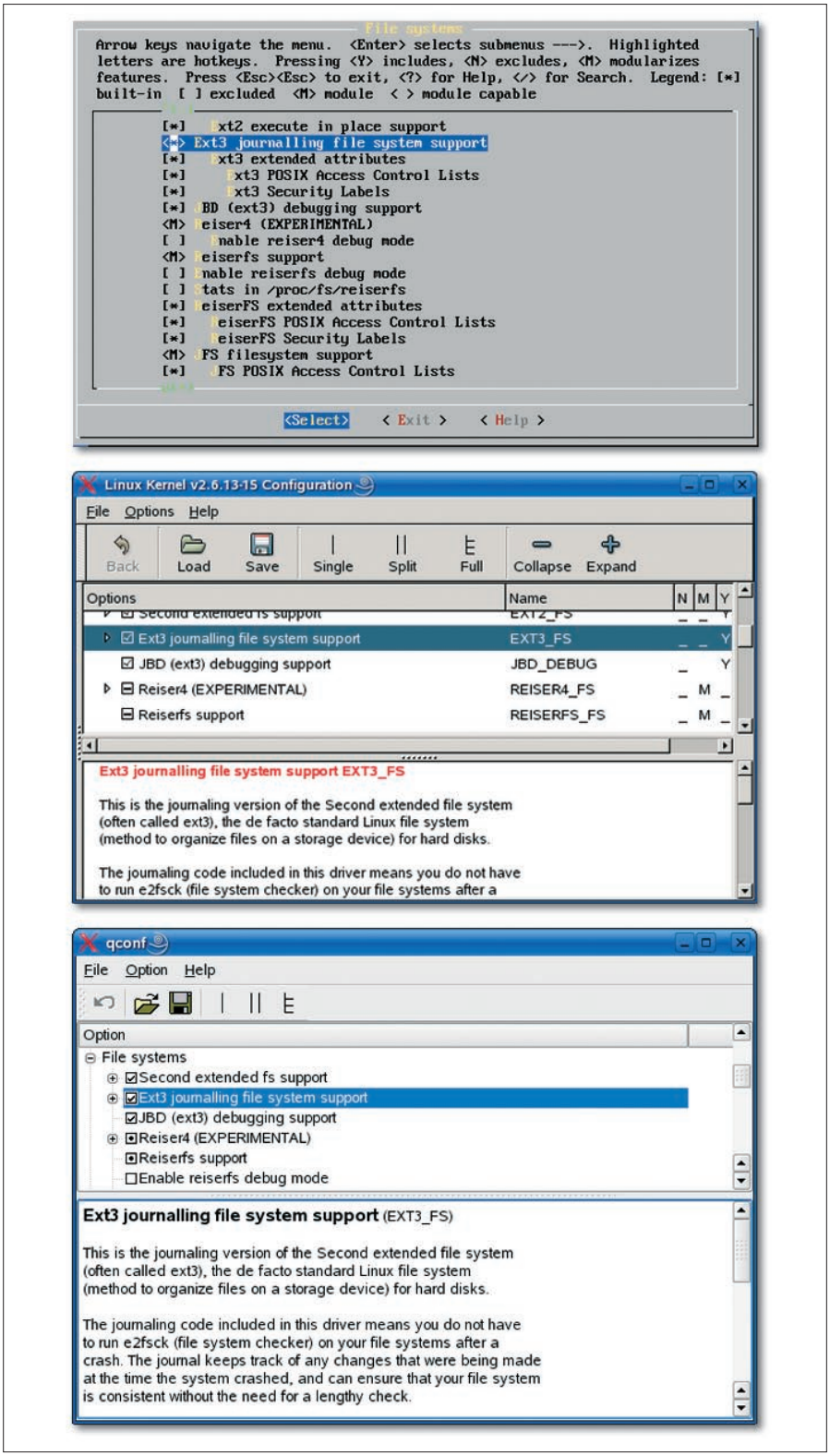

Figure 2: The same kernel configuration in various configuration tools. The driver for the Ext3 filesystem is built in, whereas Reiserfs uses a module.

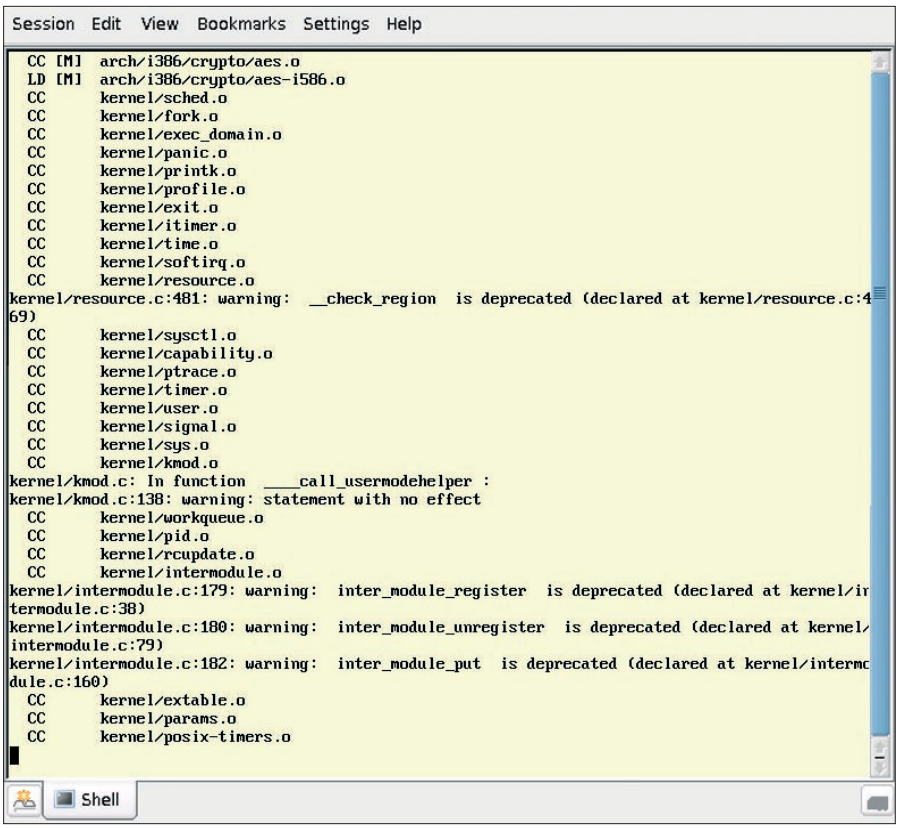

Figure 3: You can often ignore warnings during the build, but the word "error" indicates that something has gone wrong.

modules in the ramdisk. An alternative would be to build filesystem and hard disk drivers into the kernel.

Running *mkinitrd* without any parameters is typically all it takes to generate new initial ramdisks for all kernel images in */boot*. The details of *mkinitrd* vary, so reading the manpage is a good idea. If this works out, you should have a new Initrd image *initrd-[kernelversion]* in the */boot* directory.

Listing 1 shows a *menu.lst* with entries for a Ubuntu system and a Suse system. The *root* line specifies the partition with the kernel image. The *hd(0,0)* syntax stands for first hard disk, first partition, that is: *hda*. The following two lines designate the kernel image and the initial ramdisk image. The kernel image name is followed by the parameters to pass to the kernel at boot time.

Although *menu.lst* may look complex, it is actually easy to create a Grub menu entry for a new kernel. Just copy the standard entry for your distribution, and then modify the filenames for the kernel and initial ramdisk, along with the *title*. Do not remove any existing entries.

Now reboot your machine, select the new kernel from the boot prompt, and watch for error messages. The plain vanilla kernel will not give you a bootsplash screen, but you can always add the bootsplash patch [4] later.

Listing 1: menu.lst entries for Ubuntu and Suse

```
title Ubuntu, ➥
kernel 2.6.12-9-386
root      (hd0,0)
kernel    ➥
/boot/vmlinuz-2.6.12-9-386
root=/dev/hdc1 ro quiet splash
initrd /boot/➥
initrd.img-2.6.12-9-386

title SUSE LINUX 10.0
root    (hd0,8)
kernel /boot/vmlinuz root=/dev/
hdc9 vga=0x31a selinux=0 resume=/
dev/hdc3  splash=silent showopts
         initrd /boot/initrd
```

Fine Tuning

If the new kernel is happy to boot with the settings it inherits from the previous kernel, all you need to do is to modify a few settings (for example to enable drivers added by the new version). If you then recompile the kernel, only those parts of the source code affected by the changes will be rebuilt.

The "Kernel Configuration Main Menu" table gives you an overview of the critical entries at the top level of the kernel configuration menu. The menu items missing from the table are normally reserved for experts. For example, it is not a good idea to change the *Networking Support*, unless you really know a lot about the kernel's network support. Removing drivers that you do not need is not typically an issue with 2.6 kernels. In contrast to this, the version 2.4 build tended to fail because of driver interdependencies that the configuration tool had tagged as not resolved.

Never enable the debug options, as they are likely to cause major performance hits, and always create a safe copy of the *.config* file before you attempt to modify a working kernel configuration.

Up and Running

If the system works as expected, you can modify the value for the default entry in *menu.lst* to make the new kernel your standard boot option. The Kernel Build HOWTO [5] has more details on working with the Linux kernel. And the Kernelnewbies.org [6] site has a good collection of background information on kernel-related topics. ■

INFO

[1] Software Suspend2 patch (kernel patch with alternative suspend function): *http://www.suspend2.net/*

[2] Kernel changelog in "human-readable" format: *http://kernelnewbies.org/LinuxChanges*

[3] Mirrors for downloading the Kernel sources: *http://kernel.org/mirrors/countries/html/DE.html*

[4] Bootsplash patch: *http://bootsplash.org/* und *http://www.bootsplash.de/*

[5] Comprehensive HOWTO on building the Linux kernel: *http://www.digitalhermit.com/linux/Kernel-Build-HOWTO.html*

[6] Tips and information on the Linux kernel – not only for geeks: *http://kernelnewbies.org/*